SOFTWARE USER'S MANUAL FOR THE DII COE COMMON MESSAGE PROCESSOR VERSION 1.0/1.1

Solaris 2.5.1 and HP-UX 10.10

February 20, 1997

Prepared by:
Intermetrics, Inc.
615 Hope Road
Eatontown, NJ 07724

Prepared for:
Program Manager
Common Hardware Software
Ft. Monmouth, NJ 07703

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

COMMON MESSAGE PROCESSOR
SOFTWARE USER'S MANUAL

1 SCOPE

1.1 IDENTIFICATION

This user's manual provides operating procedures for the Defense Information Infrastructure (DII) Common Operating Environment (COE) message processing functional area module. It addresses requirements covering message receipt, logging, routing, storage, retrieval, generation, coordination, release, and delivery services. The COE message processing functional area software product common name is the Common Message Processor (CMP).

1.2 SYSTEM OVERVIEW

1.2.1 Common Operating Environment

The COE is intended for use by all Department of Defense (DoD) Automated Information Systems (AIS). The COE provides the infrastructure on which functional applications reside. The COE consists of an integrated architecture made up of hardware and software that provides standard support for both common and tailorable sets of functional applications software.

1.2.1.1 Common Hardware and Software.

C2 applications will be hosted on platforms defined by the DoD as Common Hardware (HW) and Software (SW) (CHS). These CHS-specified platforms are suites of processors and commercial software (e.g., operating systems, network operating systems, distributed computing software, display drivers and graphics packages, etc.). CHS provides much of the SW capability needed to support C2 applications. Support software needs not met by CHS software products are addressed through the implementation of a custom set of C2 support SW tools and are referred to as the Common Operating Environment. COE software capabilities will be developed for the following platforms (a platform is a pairing of computing hardware and an Operating System):

a. SUN Microsystems (SUN) SPARC using the SOLARIS 2.4 operating system
b. SUN Microsystems SPARC using the SOLARIS 2.51 operating system
c. Hewlett-Packard (HP) 9000 series using HP UNIX 9.07 (RISC) operating system.

d. Hewlett-Packard (HP) 9000 series using HP UNIX 10.10 (RISC) operating system.
e. Windows 3.1x and Windows NT operating systems.

1.2.1.2 Four-Layer System Model.

The DoD standard system architecture is a Four-Layer Model. Layer 1, Hardware, contains CHS processors and communications and hardware devices. Layer 2, System Software, contains the operating system and associated commercial software products (e.g., Database Management System [DBMS] X/Motif, etc.). Layer 3, Support Software, contains the software elements for 18 of the 19 COE Technical Architecture for Information Management (TAFIM) architectures (i.e., COE modules). Layer 4, Applications, contains common and/or mission-specific C2 applications. The COE area also includes a Developer's Kit.

1.2.2 Common Message Processor

The Common Message Processor (CMP) consists of objects and public operations that support the handling of incoming and outgoing message traffic. The CMP provides the COE message handler functionality in separate modules and as an integrated message handling system. The Message Processing module consists of stand alone software modules that form the core message handling functions of message receipt, logging, routing, storage, retrieval, generation, coordination, release, and delivery services. In addition, the CMP offers a CMP User Interface (CUI) from which various capabilities of the COE message handler may be accessed. This manual covers the Inbound Message Processing module in Section 3, and the Message Generation module (outbound processing) in Section 4. Section 5 contains software modules that provide additional functionality, such as operational message journal and data normalization.

1.2.2.1 Inbound Message Processing.

The Message Parser Module is a generic, table driven Message Text Format (MTF) processor that accepts formatted and unformatted messages from a communications front end, validates message format and field content, then performs data extraction as directed by the user. The Message Processor shall process all message types that conform to the MTF standard rules for message structure and content. The CMP extracts data of interest for MTF like messages and sends the extracted data to user-specified applications/processes. The parser is not considered a user application but serves as a general-purpose input interface for applications that need information from messages. Parser components may be employed as independent message processing tools performing a single function. One tool may construct message reports while another tool validates messages; still another would be used to extract information from messages.

1.2.2.2 Outbound Message Processing.

The message generation module aids in the preparation and editing of formatted text messages such as the USMTF. The system shall accommodate all messages conforming to MTF definition rules. The system must also support use of plain language address retrieval and posting to a message template through use of a user definable and maintainable table and/or use of the Defense Messaging System (DMS) X.500 global addressing scheme when available.

1.2.2.3 Message Processing Support Functions.

Normalization software converts message data from incoming messages into a format usable by the host application software. It also converts data from the format used by the host application software or databases into that of a MTF message format for outbound messages.

Typically, automatic filling of selected data fields is the first step in the message preparation process. An operator edits the partial message, inserting information not available as an automated field, and reviews the message before it is released. It is also possible to arrange a fully automatic message-generation approach, where messages are created and transmitted without any operator intervention. The CMP software module shall provide for automatic generation of formatted messages from information extracted from a database or submitted by a process. Automatic message generation means the ability to create and transmit a message without operator intervention.

1.2.2.4 The CMP Overview.

The CUI provides the user with a graphical interface from which various capabilities of the COE message handler functional area may be accessed. The CUI provides the user with a Distributed Computing Environment (DCE) based Journaling client that can be utilized with the Journaling server to provide a distributed method of monitoring all incoming and outgoing messages. Additionally, the CUI provides the user with the capability to create/edit messages interactively by accessing the CMP Message Generation/Edit capability. The CUI also provides a means to filter the incoming and outgoing messages using a dynamically updated directory through the CUI. This filtering capability allows the users to focus attention on specific messages, or types of messages, based on message attributes.

The CUI is the user's graphical interface for the management of inbound and outbound messages. The user is provided with a windows for the display and manipulation of incoming and outgoing messages. Using this interface, the user has point and click capabilities for creating, editing, and previewing all messages. There are other advanced features, e.g., the Auto-Fill functionality for message generation, the Send function for passing the message to the communications module/system, and a message header generation function.

The CUI works alongside the Outbound Message Generator (OMG), Inbound Message Processor (IMP), and Journaling software systems; hence any software that the OMG, IMP, and Journaling require is indirectly required by the CUI. The Journaling system requires that that DCE software be up and running and have an area defined for retention of data (such as a flat file or data base). The Journaling system is implemented as a DCE server while the CUI is implemented as a DCE client and uses DCE for callbacks in its graphical interface to the OMG.

2 APPLICABLE DOCUMENTS

2.1 GOVERNMENT DOCUMENTS

The following documents form a part of this user's manual to the extent specified herein. In the event of a conflict between the documents referenced herein and the contents of this manual, the contents of this manual shall be considered a superseding requirement.

2.1.1 Government Standards

The following Government standards are referenced in this document:

a. DoD 5200.28-STD, Department of Defense Trusted Computer System Evaluation Criteria

b. MIL-STD-498, Software Development and Documentation.

c. Interim MIL-STD-6040, U.S. Message Text Formatting Program, Description of U.S. Message Text Formatting Program, 1 October 1995 Implementation

2.1.2 Service-Unique Message Standards

The following service-unique message standards are referenced in this document:

a. JANAP 128, AUTODIN Operating Procedures, March 1983

b. DOI 103, DSSCS Operating Instructions

c. ACP-126, GENSER Operating Procedures

d. ACP-127, NATO Operating Procedures

e. DD173, Joint Message Form, January 1979

f. Allied Communications Publication 123 (ACP 123)

g. Allied Communications Publication 126M (ACP 126M)

h. Intelligence and Electronic Warfare (IEW) Character-Oriented Message Catalog (COMCAT)

i. United States Signals Intelligence Directive (USSID)

j. Navy Unique Message Standards

k. Air Force Unique Message Standard

l. Marine Corps Unique Message Standard (i.e., Marine Tactical System [MTS])

m. Variable Message Format (VMF) Technical Interface Design Plan for Task Force XXI, 31 July 1995.

n. Army Unique Message Standard.

2.1.3 Other Government Documents

The following Government documents are referenced in this document:

a. Software Requirements Specification for the Joint Message Analysis Processing System, 14 September 1994

b. Joint Message Analysis Processing System (JMAPS) User's Manual

c. Joint Message Analysis Processing System (JMAPS) Programmer's Manual

d. Joint Message Preparation System (JMPS) User's Manual

e. User Interface Specification for the Global Command and Control System (GCCS)

f. ACCS-A1-302-001B, Common ATCCS Support Software (CASS) Segment Specification (CSS) Draft, 14 November 1994

g. ACCS-A3-500-004, Army Command and Control System Message Catalog, 28 May 1993

h. ACCS-A3-500-005, Message Format Definition Database Specification, 21 July 1993

i. LL-500-04-03, GCCS Common Operating Environment Baseline, DISA, November 1994.

2.2 NON-GOVERNMENT DOCUMENTS

The following non-Government documents are referenced in this document:

a. UNIX applicable documents

b. SUN SOLARIS Version 2.3 or later

c. SUN Operating System Version 4.1.2 or later

d. HP UNIX Operating System Version 8.0 or later

e. Open Systems Foundation (OSF) Distributed Computing Environment (DCE)

f. TRANSARC Distributed Computing Environment (DCE)

f. Cherinka, Robert D. and Collins, W.J., "The Joint Message Analysis Processing System (JMAPS) Development Environment User Guide," Draft, MITRE Corporation, Bedford, Massachusetts

g. Cherinka, Robert D. and Riffe, Annette S.H., "The Joint Message Analysis Processing System (JMAPS) Version 3.3, Release Notes," D075-LL-075, MITRE Corporation, Bedford, Massachusetts

h. Cherinka, Robert D.; Malloy, Mary A.; and Renner, Dr. Scott A., "The Joint Message Analysis Processing System (JMAPS) Users Manual, Version 3.3," WP 94B0000076, MITRE Corporation, Bedford, Massachusetts

i. Miller, Dr. Robert W. and Scarano, Mr. James G., "The Design and Implementation of the Phase III Joint Message Analysis Processing System (JMAPS)," MTR 92B0000097, March 1993, MITRE Corporation, Bedford, Massachusetts

j. Riffe, Annette S.H., "JMAPS Enhancements to Enable the Processing of U.S. Navy OTH Gold Message Text Formats," WP 93B0000135, May 1993, MITRE Corporation, Bedford, Massachusetts.

3 INBOUND MESSAGE PROCESSING

(NOTE: SECTION 3 IS NOT FOR DII END USER)

3.1 COMMON MESSAGE PROCESSOR OVERVIEW

This section describes the procedures to use the stand alone Message Processor module of the CMP. The Message Processor is a set of computer programs designed to process MTF like messages. The CMP's processor receives messages from the communication module and extracts the data of interest, sending it to user applications such as databases, applications, and report generators. The processor is not part of these applications and does not include them; rather, it serves as a general purpose input interface for applications that need information from messages.

3.2 SECTION OVERVIEW

This document is a user's guide for the CMP Inbound Message Processor functions. It explains how to install and perform inbound message processing.

Knowledge of some basic computer functions and the UNIX operating system is assumed (i.e., how to give commands to the shell, view and edit a file, change directories, etc.). The on-site

system administrator must be familiar with several details of UNIX administration, including building and installing a new kernel, creating user accounts and user groups, extracting files from tape, etc.

Familiarity with the MTF standards is assumed. MIL-STD-6040 provides an introduction regarding this area.

This section contains several examples of how to use the Common Message Processor. These examples contain a few special terms and a few special type styles, as explained in the following list:

a. Standard type represents computer output.

b. **Bold** type represents something that must be typed exactly. For example, if prompted to type **mtfx file.1**, all 11 characters must be typed exactly as printed in the manual, including the space between **mtfx** and **file.1**.

c. *Italic* type is used for emphasis or to introduce a new term. It may also represent a placeholder for something that must be provided. For example, if asked to type *filename*, then the actual name of a file instead of the eight characters printed in the manual should be typed.

d. CAPITALS represent the name of a key on the keyboard; for example, CTRL, SHIFT, ENTER, etc.

e. KEY1-KEY2 means to hold down the first key while pressing the second key. For example, if asked to press CTRL-Z, hold down the CTRL key, press the Z key, then release both keys.

3.3 EXECUTION PROCEDURES

3.3.1 Message Processing

The CMP processor is made up of modular, general purpose message processing software that extracts data from received messages. Its design allows it to be easily interfaced with existing and developing systems. These systems can use the processor to satisfy message processing and preparation requirements instead of developing an internal, independent capability. Each system needs only to know the information that it must extract from incoming messages; this module will handle the extraction and the validation of the data extracted.

The CMP processor may be employed in several different ways, depending on the needs of the user. It may be operated as a central message processor, serving a number of client systems on a network or function as an integrated component of a system. Finally, the processor may be employed as a set of independent tools.

3.3.1.1 USMTF Message Processing.

The USMTF standard is defined in MIL-STD-6040. It specifies over 200 text message formats for joint and tactical information exchange. These Message Text Format (MTF) messages are designed to be human readable and machine processable.

3.3.1.2 Variable Message Format.

The Variable Message Format (VMF) standard is defined in the Technical Interface Design Plan (TIDP) for Task Force XXI, dated 31 July 1995. It specifies message formats for Bit Oriented Messages (BOM) for information exchange. These messages are designed to be machine processable and are not human readable.

The CMP contains software capable of converting a VMF (BOM) message into a Character oriented Message (COM) resembling a MTF message and processes the message as if it were transmitted as COM. The Marine Corps Tactical System (MTS) messages are a special subset of the VMF message suite and are a sub-set of the VMF messages processed by the CMP.

3.3.1.3 The Requirement for MTF Message Processing.

The Joint Chiefs of Staff (JCS) require the use of MTF messages in all record message traffic between services in joint and combined operations.

It is clear that many existing and future Automated Information Systems (AIS), to include C3I systems, must handle MTF messages to satisfy interoperability requirements and exchange of command and control data for machine processing and rapid integration of bulk data into existing data bases. These systems will have to construct outgoing MTF messages in order to supply information to other systems, and will have to receive and process incoming MTF messages in order to obtain information. MTF message processing means a great deal more than simply routing messages to the proper destinations. Automation requires extraction of specific data from messages for internal use. AIS systems must be able to extract the specific data elements they require from the messages they receive. If the wrong elements are extracted, the operation/task(s) will fail. (For example, a mission-planning system that mistakenly extracts a refueling rendezvous instead of a target location will not produce a valid plan.) To avoid these mistakes, it is necessary to determine the correspondence between the message text and the structure of the message format. This is known as parsing the message, which ensures that a system receives valid and correct data.

3.3.2 CMP: A Central Message Processing Service

The CMP may be employed as a central message processor, providing services to many different client programs across a local area network. In this arrangement, the clients need never handle

the complete message. Instead, they use the processor as a "black box" interface that preprocesses incoming messages and constructs outgoing messages on their behalf.

The CMP and the client programs may execute on the same machine, or on separate machines on the same network. For example, a communications processor may handle the actual transmission and receipt of messages from other locations while a message handler routes or sends a copy of each message to local operators and systems. Also, a message release authority may make the final decision to transfer an outgoing message to the communications processor for transmission. It is possible to define parsing requirements in real time but normally before operation begins each client system will specify the information it requires from incoming messages. For example, one system might request the time and location fields for each contact reported in each TACELINT message. Another system might request the details of refueling missions tasked to a particular wing from ATOCONF messages.

During operations, incoming messages are received by the communications processor. The local message router or message handler sends a copy of each message to the message parser. (If there is no local message router, the parser can be connected directly to the communications processor, and can also route entire messages to local operators or systems.) The CMP processes each message as it arrive, and sends to each client the parts of the message the client has asked to receive. The client systems never have to read the incoming messages. Instead, they read just the fields that they require, broken out of the message and placed in a system specified input format. A client can use this input to modify a situation display, prepare a report, update a database, etc. If desired, the input supplied by the parser may be reviewed by an analyst before being processed by the client system.

The CMP can test each message for conformance with current message standards. If the content or structure of a message deviates from the standard, then the integrity of its information is in question. Client applications can use this capability as a partial validation of their input.

The CMP can store messages it receives. Users can then write interactive queries to the message contained in that repository, expressing requests such as "show me all the messages that assigned refueling missions to this wing in the last two days." The Query Language used is based on Structured Query Language (SQL), the *de facto* standard language used by many commercial databases.

Outgoing messages may be generated in one of two ways. An operator of a client system may invoke the interactive message editor and manually construct the message (see Message Generation, paragraph 4), or the client program may generate the message data and send it to the parser (see MDLMAP, section 5), which transforms the data into the message text format. A combined approach is possible, in which the message is partially formed by the processor and then manually completed using the message editor. The constructed messages may be reviewed by a message release authority, an automated or human interactive process, before they are actually transferred to the communications center for transmission. A major advantage of this modular architecture is that the client programs do not have to include their own message processing

component. This makes the client programs easier to design, build, and maintain. There are also efficiency advantages: because it acts as a central message processor, each arriving message is processed only once, instead of once by each client program. The CMP makes a further improvement in efficiency by doing only the processing required to satisfy the combined requirements of its clients; if no client needs information from a particular message type, then those messages are not further processed.

3.3.3 CMP: A Set of Integrated Message Processing Components

The CMP may be employed as a set of message processing components that are integrated into some other system. The CMP offers an integrated Message Handling System with a CMP user Interface described in Section 4.9. It uses the CMP as one of its components to perform message processing tasks. The difference between this and the central server arrangement is primarily one of organization and control. In the previous scenario, the parser received all incoming messages and then forwarded information extracted from these messages to its clients. In this scenario, the parent system receives incoming messages, and then uses the CMP module as a sub process to validate these messages and extract information from them.

This employment of the CMP is appropriate when the parent system has message processing requirements that cannot be completely satisfied by the CMP. In this case, the parent system must directly exchange messages with the local message router, instead of indirectly via the CMP. The primary advantage of using the CMP is retained because the parent system does not have to re-implement the capabilities of the software. However, if several systems on the network adopt this approach, then the efficiency advantage of a central message processor is lost.

3.3.4 CMP: A Set of Stand Alone Message Processing Tools

The CMP may be employed as a set of independent message processing tools. Each tool performs a single function: one tool constructs messages, another validates messages, a third extracts information from messages, etc. These tools are typically used directly by an application, but can also be configured to run as a background process.

This employment of all modules of the CMP is appropriate when its full functionality is not required. For example, if the only requirement is to manually prepare outgoing messages, then there is no need for the data extraction and routing capabilities. These simple requirements are frequently encountered on small, single-user personal computers or workstations. The independent tools are suited to this environment.

The message editor component is available as a stand-alone software module (see Section 4, Message Generation).

3.4 CMP ARCHITECTURE AND EXECUTION MODEL

The CMP message-processing server is composed of several subsystems that cooperate to validate incoming messages, extract data from messages, and route extracted data to external, client systems. This section describes the architecture of the CMP server, and then traces the passage of an input message through the system. (The stand-alone CMP tools are not a part of the server and are described in Section 3.10.)

3.4.1 Major Components of the CMP

The message processing server has three major interfaces. They are the Message Input, Application Interface, and Operational Message Journal.

a. The Message Input/Output (MIO) is the initial point of entry for incoming messages. The MIO reads new messages that have been placed in an input directory by an external agent.

b. The Applications Interface (AI) manages the interface between the parser and the client applications. The AI receives messages, performs the validation and data extraction required by the client systems, and then sends the extracted information to the clients.

c. The Operational Message Journal stores messages received/generated by the system/user.

Currently, the UI allows an authorized user to modify the system configuration. It also allows interactive access to the validation, correction, and data extraction capabilities. The UI communicates with the message processing server by accessing the data storage repository components through shared memory.

The Message Generation components also execute as a separate process. They support the composition, editing, validation, and correction of messages (see paragraph 4).

3.4.2 Operation of the CMP

The CMP server may execute in the background as a daemon process. The server goes through three distinct phases of execution: startup, message processing, and termination. In addition to background message processing, the CMP also responds to interactive user requests through the user interface.

3.4.2.1 Startup/Initialization Phase.

When the CMP is started, it first reads a master configuration file called the *.MAPSconfig* file. This file tells the CMP where to look for new messages, where to store old messages, where to find other configuration information, etc. The CMP then searches the configuration directory for files with a name ending in the four characters *.ccf*; these are *client configuration files*, which describe the information required by client applications. At this point, the CMP is completely initialized and ready to receive incoming messages (see Section 3.9.2 for more information about the .MAPSconfig file, and Section 3.8.1 for more about the client configuration files).

This initial startup is called the *coldstart* process. It is also possible to restart the CMP from a state file created during a previous execution; this is called a *warmstart*. During a warmstart, the CMP ignores the client configuration files; configuration information comes from the saved state file instead.

3.4.2.2 Message Processing Phase.

The CMP expects new messages to arrive as files in a directory called the *mio* directory. The location of this directory is specified in the .MAPSconfig file. The MIO software repeatedly scans this directory, searching for new files with a filename ending in the three characters *.in*. These files contain messages or parts of messages. When the MIO discovers a new message file, it removes the file from the *mio* directory, and then sends the file contents to the parser for further processing.

The parser module determines how each message should be handled. There are several possibilities, as follows:

a. If the file does not contain a recognizable message, then it is dumped to a directory containing invalid input files, and this event is recorded in the audit log.
b. If the file is one part of a sectioned message, then it is retained until all sections arrive or the user directs that processing of available sections is to occur.
c. If the file contains an entire message, or is the last missing piece of a sectioned message, then the entire message is forwarded to the AI for further processing.

The parser module validates and parses the message as it executes the queries. The parser does the minimum amount of parsing required; that is, if no query references a particular part of the message, then that part of the message will not be parsed. This is known as *query-directed parsing*. Each client specifies the amount of message validation it requires, and the CMP does the minimum amount of validation required.

After handling all of the routing table entries for the new message, the CMP consults the application Trigger Table ATTable to determine whether this message should be saved in the operational journal. The messages in the journal can be inspected, queried, and corrected interactively through the CUI.

At this point, the processing of this message is complete. Control returns to the MIO subsystem, which repeats the process by looking for a new message file.

3.4.2.3 Shutdown/Termination Phase.

When the CMP is told to terminate, it first writes a *snapshot* file describing the current state. This file includes the messages in the message journal, the contents of the ATTable, and, in general, everything needed to resume execution of the server in its present state. This state file will be read by the CMP during a subsequent warmstart operation.

3.4.2.4 Interactive Processing.

While the in-bound message processor is executing in the background, the UI permits operator access to the messages saved in the operational message journal and to the client configuration information in the ATTables. The UI permits an operator to perform the following actions:

a. List the messages saved in the journal
b. View the text of any message saved in the journal
c. View an error report listing all validation errors in a message
d. Correct and edit a message
e. Resubmit a message, causing the CMP to process it as if it had newly arrived
f. View the output of a query executed on every message in the journal

g. Allows an authorized user to archive, purge, or selectively remove messages from the message database

h. Edit the client configuration information in ATTable.

It is possible to: (1) run the message processing server without ever running the UI; (2) start and stop the UI several times while the server is in the message processing phase of execution; or, (3) run several UI processes concurrently, allowing multiple operators to access the (single) message-processing server.

3.5 MESSAGE PROCESSOR

This section contains examples of the processor in use, describing a hypothetical client application and showing how to specify the information it requires. It also gives examples of the interactive operations available through the user interface. This section also includes a brief overview of the query language, client configuration files, and the processor-to-client interface. (For more specific registration information and procedures see Appendix A.)

3.5.1 Interactive Capabilities

This section covers some tasks that can be performed with the current CMP UI: examining messages in the journal, running interactive queries, and correcting messages which contain errors.

3.5.1.1 Examining Messages.

The processor can be configured to save certain types of messages in the message journal. These messages can be listed and examined through the Message Journal window in the UI. Messages are identified by their type (the message ID field), the date-time group, and the originator. These items are included in the header lines sent with each query output to the client applications. For example, the header lines in Figure 3-9 indicate that the information in the query output was extracted from the message identified by the triple "TACELINT, 011200Z APR 93, RHDIAAA." To find this message, we look for these strings in the display. We can then examine the text of the messages saved. To examine the TACELINT message mentioned above, we first select it by clicking on the third line in the display. Next, we press the View button. This creates the Message Journal (VIEW) window. When we are finished looking at this message, we press the Cancel button to dismiss the window.

We can also examine an "exploded" version of a message, in which each field and each coded value is labeled with a descriptive string. This is called *report format*. While messages in this format are much longer than the cryptic versions, they are also much easier to understand. The report format allows users to read a message even when they are not familiar with the message format.

To examine our sample TACELINT message in report format, we first select it as before. Then we press the Report button.

The message journal allows us to selectively display messages, choosing by message type, date-time group, and originator. We can also save, print, and delete messages here. These features are covered in Section 3.7.3.

3.5.2 Interactive Queries. ((Not currently implemented))

With the user interface we can compose a query and execute it against the messages journal/repository "on the fly." It is not necessary to change the configuration files. This interactive query facility permits us to treat the messages in the journal as a message database. We can write useful queries to extract information from a single message, or from the entire message database. This section will discuss both types of interactive query.

Like many ATOCONF messages, this message is several thousand lines long and contains tasking for many different units. Suppose we want to review the portion of the message which contains tasking for a particular unit; for example, the 4th Fighter Wing. To do this manually, we must review the entire message, because a unit's tasks do not have to be grouped in a single place. Instead, we will use the processor to extract the particular segments. To do this, we must first compose and then execute a new query.

To compose a query, we follow the same steps outlined in Section 3.5.1.2. The name of our new query is "4fw." The text of this query is as follows:

SELECT TASKUNIT.1,[MSNDAT]

FROM MSGID="ATOCONF"

WHERE TASKUNIT.1="4FW"

FORMAT TEXT;

There are a few new features of the query language present in this query:

a. [MSNDAT] selects the entire MSNDAT segment, which contains all the information for a particular air mission (see Section 3.6.1.1).

b. By default, interactive queries are executed on every message which meets the FROM criteria in the database. The FROM clause means that this query will only be executed on ATOCONF messages (see Section 3.6.1.3).

c. FORMAT TEXT means that the query will output the exact text of the selected sets and segments, complete with field separators (see Section 3.6.1.2).

When we finish adding the new query, we will see it appear in the Queries window. Instead of closing this window, we now select the *4fw* query and then press the Execute button. This will execute the query on the only message in the repository which matches the FROM clause. (If there were two or more ATOCONF messages, we would need to write a longer FROM clause to

pick messages by type and date, and possible originator.) The query output appears in the Query Result (VIEW) window.

3.5.3 Correcting an Invalid Message.

Client systems may specify that they wish to be notified if the data sent by the processor was extracted from an invalid message. The processor sends this notification in the .ERRORS line in the output wrapper. There are many things a client system might do when it receives data extracted from an invalid message; for example, it could accept it or silently discard it. A client system might request that the invalid message be corrected and reprocessed. This section shows how to use the UI to do this.

Suppose the processor has been configured with the *isum.ccf* file, see Appendix A, and an invalid TACELINT message arrives. It will execute the *isum-01* query and send the results to ISUM. Because this query includes a VALIDATE ALL clause, the processor will report any validation error occurring anywhere in the message. The data sent to the ISUM program will appear as follows:

MSGID TACELINT

FROM RHDIAAA
TO DHDIAZZ/SYJ
DTG011422Z APR 93
ERRORS 3 MESSAGE,EXTRACTED
"24492","911945Z","LS:435244N0751836W","SPRUANCE","CUSHING","F","SHIP"
END

ISUM sees a non-zero value in the ERRORS line in the output wrapper and recognizes that this data comes from an invalid message. ISUM will not update its database with this information. Instead, ISUM asks its operator to correct and reprocess the invalid message. ISUM uses the values from the other lines in the output wrapper to identify the message, telling the operator to look for "TACELINT, 011422Z APR93, RHDIAAA" in the message repository.

Acting now as the ISUM operator, we switch to the UI, choose Windows Message Journal to open the window, and select the line matching the message specified by ISUM. Next, we press the Errors button. This creates the window which contains a listing of the validation errors found in the message.

The error report indicates a problem with the second field of the SOI set. To discover the meaning of the SOI.2 field, we examine the message in report format (by pressing the Report button). The following represents some of the resulting output:

SOI

TARGET SIGNAL IDENTIFIER::	24492
DETECTION TIME::	911945Z
TIME LOST::	011959Z
ELINT NOTATION::	EHIZZ
EMITTER DESIGNATION::	LOUDMOUTH
AIR DEFENSE DISTRICT::	787

The SOI.2 field is the "detection time" field. The value of this field tells up that this signal was first detected at 1945 hours on the 91st of the month. The processor recognizes that this is not a valid date value, and therefore reported an error for this field.

With a little detective work, we can guess the correct value for the SOI.2 field. The "time lost" field tells that the signal was lost at 1959 hours on the first of the month. It is very likely that this signal was detected and lost on the same day. The first character of SOI.2 is probably a typographical error; it should be a "0" instead of a "9."

To change the SOI.2 field, we press the Correct button. This sends the selected message to the message editor. The parser displays the text of the message, replacing the incorrect field with underscore characters to show where we must make our correction. Section 4 describes the editor window and how to correct messages.

Using the CMP Message Generator, enter the correct contents ("011945Z") into the SET SOI, field 2.

When we choose the Exit menu selection, the windows will vanish, returning us to the Message Journal window. To verify that the message has been corrected, press the View button or the Errors button. The next step is to reprocess the message. To do this, we press the Submit button. The processor then processes the corrected message as if it had just been received. It again executes the *isum-01* query and sends the results to ISUM. This time, the data sent to ISUM appears as follows:

MSGID TACELINT

FROM RHDIAAA

TO DHDIAZZ/SYJ

DTG 011422Z APR 93

ERRORS 0

"24492","011945Z","LS:435244N0751836W","SPRUANCE","CUSHING","F","SHIP"

END

ISUM will accept this data, since the output wrapper indicates that no validation errors were found in the source message.

3.6 QUERY LANGUAGE REFERENCE

The query language is a language for extracting information from messages and/or files/data bases. Queries written in SQL specify the information to be retrieved and the output format used to view the retrieved data. A query specific enough to identify a single message, in which case it acts as an information filter. A query can also be applied to a collection of messages in the message journal and acts as a database retrieval query.

SQL is a well known data retrieval language for relational databases. SQL has been extended to deal with structured messages, instead of totally flat relational tables. A design goal of the language is that anyone who is familiar with SQL and with the general structure of messages should find queries easy to formulate.

This section describes the query language. The first part describes the structure of queries and the meaning of the keywords in the language. The second part describes an extension to SQL queries, called shells, which permits concatenation of multiple queries and substitution of query output strings into a text template. Shells provide both a simple report generator capability and a way to generate SQL statements to update a database with information extracted by a query.

3.6.1 Queries

There are three keywords used in SQL:

a. A *select* clause is used to specify the particular data elements to be retrieved from one or more messages.

b. A *from* clause is used to describe the message domain; that is, those messages from which data is to be extracted.

c. A *where* clause is used to restrict both the message domain and the data retrieved by specifying conditions that must be satisfied by the message contents.

CMP has three additional keywords not found in SQL. These were added to handle the structural features found in operational messages but not found in relational databases.

a. A *format* clause specifies the output form of the extracted data items

b. A *within* clause restricts data extraction to the fields contained in the specified message segments.

c. A *validate* clause specifies the amount of message validation that is to be performed during data retrieval.

A query contains one or more clauses, each beginning with one of these six keywords. Every query ends with a semicolon. Queries are not sensitive to the difference between uppercase and lowercase letters.

The rest of this section discusses the meanings of the clauses in a query.

3.6.1.1 The SELECT Clause.

The SELECT clause describes the data elements that will be retrieved by the query. Every query must have exactly one SELECT clause. This clause consists of a list of *data element specifiers*. These specifiers correspond to parts of the message format: segments, sets, and fields. There is a data element specifier for every element. As we describe these specifiers, we will refer to the sample LOCATOR message in Figure 3-1.

OPER/BIG FLOATER//

MSGID/LOCATOR/RHDITAC/0401004/APR//
REF/A/ORDER/CTF122/011630ZAPR93/010200/NOTAL/AAS/AAT/AAZ//
REF/Z/DOC/AF-II/312359Z/3112009/PASEP/ABC/ADD//
QUEWORD/RED//
SUB/075/GRYPHON/LR12/CERT/UR/NUC/DELTA II/HIGH/PHT//
TMPOS/010735Z/4223N08640W/170T/12KTS/REGAIN/SURF/VISUAL//
TMPOS/010855Z/4224N08644W/175T/12KTS/HOLDING/SURF/VISUAL//
SUB/076/RED OCTOBER/LR12/CERT/UR/NUC/DELTA II/HIGH/PHT//
TMPOS/010822Z/4324N08144W/140T/15KTS/REGAIN/SURF/VISUAL//
NAVAL/076/USS HARLAN COUNTY/LR02/CL-MOD KASHIN/DD/UR/NPH//
BRNG/010645Z/4259N07548W/165T//
TMPOS/010635Z/4228N08633W/180T/14KTS/LOST/AIR/ACSONO//
ATTACK/MK46/UNSUCCESSFUL/VISUAL/5NM/-/180T//
TMPOS/010640Z/4228N08635W/180T/14KTS/GAINED/AIR/ACSONO//

Figure 3-1. Sample LOCATOR Message

A set is denoted by its unique set identifier. The specifier "MSGID," applied to the message in Figure 3-1, would retrieve the entire set, producing the following result:

MSGID/LOCATOR/RHDITAC/0401004/APR//

A set specifier retrieves every matching set from the message domain. The specifier "TMPOS" applied to the example message would retrieve the following five sets:

TMPOS/010735Z/4223N08640W/170T/12KTS/REGAIN/SURF/VISUAL//

TMPOS/010855Z/4224N08644W/175T/12KTS/HOLDING/SURF/VISUAL//

TMPOS/010822Z/4324N08144W/140T/15KTS/REGAIN/SURF/VISUAL//

TMPOS/010635Z/4228N08633W/180T/14KTS/LOST/AIR/ACSONO//

TMPOS/010640Z/4228N08635W/180T/14KTS/GAINED/AIR/ACSONO//

A set can also be denoted by its *presentation number*, or *pnum*. A presentation number is an integer value assigned to each set during message analysis, indicating the set's position in the structure of the message type. A set specifier using a pnum is written as an integer in parenthesis; for example, "(31)." This kind of set specifier retrieves only those sets in the corresponding position. Pnums help to resolve ambiguities that arise when the same set appears in different positions in a message type.

To determine the presentation number for a particular set in a message, you must consult the entry for the message in MIL-STD-6040, *Catalog of USMTFs*. The entries show the sequence of sets in the message type. The presentation number for each set is given in the column labeled "SEQ." Figure 3-3 shows part of the entry for the LOCATOR message type. The TMPOS set appears 21 times in this message type, at presentation numbers 19, 31, 34, 36, etc. Suppose that analysis of the LOCATOR message in Figure 3-2 shows that the five TMPOS sets in the message occurred at pnums 31, 34, 31, 55, and 60, respectively. Then the set specifier "(31)" would retrieve only the first and third TMPOS sets. We can assign numbers to the fields in a set, numbering sequentially from left to right. A particular field is denoted by the name of the set in which it appears, a period ("."), and the ordinal number of the field within that set. For example, when applied to the sample LOCATOR message:

MSGID.1 retrieves LOCATOR

MSGID.2 retrieves RHDITAC

MSGID.3 retrieves 0401004

MSGID.4 retrieves APR

Note that the equivalent specifiers for these four fields using presentation numbers instead of set identifiers would be "(3).1," "(3).2," "(3).3," and "(3).4," since Figure 3-17 indicates that the MSGID set is assigned presentation number 3.

As with set specifiers, a field specifier retrieves every matching field in the message domain. For example, the specifier "TMPOS.5" retrieves five fields: "REGAIN," "HOLDING," "REGAIN," "LOST," and "GAINED."

Certain fields are repeatable; that is, they may appear an arbitrary number of times in a set. If such a field is specified, each repetition is retrieved. For example, the seventh field in the REF set is repeatable. The specifier for this field, "REF.7," retrieves five data items: "AAS," "AAT," "AAZ," "ABC," and "ADD."

A segment is denoted by enclosing the identifier of its first set in square brackets. The specifier "[ATTACK]" retrieves every segment beginning with an ATTACK set. When applied to the example message, it retrieves:

ATTACK/MK46/UNSUCCESSFUL/VISUAL/5NM/-/180T//

TMPOS/010640Z/4228N08635W/180T/14KTS/GAINED/AIR/ACSONO//

A segment specifier also retrieves every matching segment in the message domain. In our example, the specifier "[SUB]" retrieves the following two segments, containing five sets in all:

SUB/075/GRYPHON/LR12/CERT/UR/NUC/DELTA II/HIGH/PHT//

TMPOS/010735Z/4223N08640W/170T/12KTS/REGAIN/SURF/VISUAL//

TMPOS/010855Z/4224N08644W/175T/12KTS/HOLDING/SURF/VISUAL//

SUB/076/RED OCTOBER/LR12/CERT/UR/NUC/DELTA II/HIGH/PHT//

TMPOS/010822Z/4324N08144W/140T/15KTS/REGAIN/SURF/VISUAL//

Certain segments can be initialized with any one of a choice of two or more sets; these are called *alternate initial sets*. It is important to remember that a special interpretation of segment specifiers applies in such cases.

a. If the segment is referenced using the initial set having the lowest presentation number from among the alternates, then the segment specifier retrieves *all* occurrences of that segment, regardless of which alternate initial set was actually used.

b. If the segment is referenced using any initial set other than the one with the lowest pnum, then the segment specifier retrieves *only* those occurrences of the segment which used that specific alternate initial set.

The example LOCATOR message does not contain any segments with alternate initial sets.

Segment specifiers may be used to restrict the domain to which a set or field specifier is applied. To refer to TMPOS sets only when they appear in NAVAL segments, we write "[NAVAL]TMPOS." This retrieves just two of the five TMPOS sets in the message:

TMPOS/010635Z/4228N08633W/180T/14KTS/LOST/AIR/ACSONO//

TMPOS/010640Z/4228N08635W/180T/14KTS/GAINED/AIR/ACSONO//

We can also specify the first field in TMPOS sets in NAVAL segments by writing "[NAVAL]TMPOS.1." This retrieves "010635Z" and "010640Z." Once again, pnums can be used instead of set identifiers to formulate segment specifiers. Figure 3-2 shows that the first field in TMPOS sets in the ATTACK segment that are nested within the SUB segment can be denoted [24][35]TMPOS.1 by using the corresponding set pnums.

(U) INDEX REFERENCE NUMBER :C325 STATUS : AGREED DATE : 01-DEC-1985
MTF IDENTIFIER :LOCATOR
MESSAGE TEXT FORMAT NAME :MARITIME FORCE LOCATOR
FUNCTION OR PURPOSE :THE LOCATOR IS USED TO REPORT SURFACE OR SUBSURFACE, AIR, OR SPECIAL INTEREST UNITS OPERATING IN THE MARITIME ENVIRONMENT

SPONSORS :
RELATED DOCUMENTS :
MESSAGE TEXT FORMAT :

SEG	RPT	OCC	SETID	SEQ	FIELD OCCURRENCE	SET FORMAT NAME
		(C)	EXER	1	/M/O//	EXERCISE IDENTIFICATION
		(O)	OPER	2	/M/O/O/O//	OPERATION IDENTIFICATION DATA
		(M)	MSGID	3	/M/M/O/O/O/O//	MESSAGE IDENTIFICATION
		*(O)	REF	4	/M/M/M/M/O/O/*O//	REFERENCE
		(C)	AMPN	5	/M//	AMPLIFICATION
		(C)	NARR	6	/M//	NARRATIVE INFORMATION

[Some sets have been omitted]

C		(M)	SUB	24	/M/M/M/M/M/M/O/O/O/O/O//	SUBMARINE CONTACT
[		(O)	ASSOC	25	/*M//	ASSOCIATION
[		(C)	ELLIPSE	26	/M/M/M/M//	ELLIPSE
[		(C)	CIRC	27	/M/*//	CIRCULAR AREA
[		(C)	AREA	28	/*M//	AREA
[		(C)	TIMEVENT	29	/M/M/M/M/M/O/O//	TIME INFORMATION
[	*	(O)	BRNG	30	/M/M/M/O/O/O//	BEARING
[	*	(C)	TMPOS	31	/M/M/M/M/M/M/O/O/O//	TIME AND POSITION
[	*	(O)	RELPO	32	/M/M/M/M/M/O/O//	RELATIVE TO OWN POSITION
[	*	(O)	REACT	33	/M/M/M/M/M/O//	REACTION
[		(C)	TMPOS	34	/M/M/M/M/M/M/O/O/O//	TIME AND POSITION
[O		(M)	ATTACK	35	/M/M/O/O/O/O//	ATTACK
[[		(M)	TMPOS	36	/M/M/M/M/M/M/O/O/O//	TIME AND POSITION
[						END OF SEGMENT

[Some sets have been omitted]

			DECL		/M//	MESSAGE DOWNGRADING OR DECLASSIFICATION DATA

Figure 3-2. Description of LOCATOR Message Format

The asterisk character is a **wildcard** that can replace a field number or set identifier in a specifier. It has the following meanings:

*	every set
[*]TMPOS	the TMPOS sets in every segment
[SUB]*	every set in a SUB segment (equivalent:[SUB])
TMPOS.*	every field in a TMPOS set (equivalent: TMPOS)

There is one special function that can be applied to data elements: the NAME function. When applied to a message or segment, the NAME function yields the set identifier of the initial set of the message or segment enclosed in square brackets. When applied to a set, NAME produces its set identifier. When applied to a field, the NAME function returns the FFIRN/FUDN that describes the field content's format. For example, referring to the sample message:

a. NAME([ATTACK]) or NAME([20]) is [ATTACK]
b. NAME(TMPOS) or NAME(21) is TMPOS
c. NAME(TMPOS.1) or NAME(21.1) is 1131/2

The NAME function is primarily used when one wishes to determine which of several alternate formats has been used for a particular field in the message.

It is possible to have any number of data element specifiers in a SELECT clause. Effectively, an internal table of the selected data items in the message is built, with a separate column for each data element specifier. However, this table is not what it prints as the query output. The query output is controlled by the FORMAT clause, described in the next section.

3.6.1.2 The FORMAT Clause.

The FORMAT clause controls the output of the data items extracted from the message by the SELECT clause. There are four output formats available:

a. *text* format, which presents the extracted data items as a "flat file"
b. *table* format, which presents the data as a relational table
c. *errors* format, which creates a listing of validation errors in the message
d. *report* format, which creates an annotated listing of a message.

The FORMAT clause is optional; it may appear no more than once in a query. By default, query results are presented in text format.

3.6.1.2.1 Text Format.

In text format, the processor prints the extracted data items as a flat American Standard Code for Information Interchange (ASCII) file. All of the items extracted by the first data element specifier are printed, one per line; then all of the items extracted by the second specifier, and so forth. It

prints a blank line at the end of each group of data items. Within a group, data items are printed in the order of appearance in the message. Selected data items that appear more than once in the message are printed more than once in the output.

For example, suppose we execute the following query on the message in Figure 3-3.

SELECT MSNDAT.1, TGTLOC.1, TGTLOC.2

FROM MSGID="ATOCONF"

FORMAT TEXT;

The output of this query would be as follows:

1201I

1205H

[blank line]

241010Z

241015Z

241020Z

[blank line]

241020Z

241035Z

241040Z

[blank line]

There are three groups of data items in the output, one per data element selector in the query.

OPER/PARSER DEMO//

MSGID/ATOCONF/USCENTAF-CMBT PLANS/040101/APR//

PERID/312320Z/TO:010501Z//

AIRTASK/UNIT TASKING//

TASKUNIT/C41//

MSNDAT/1201I/JAN/EAGLE 01/3A6E/INT/-/BEST/-/20000/32001//

TGTLOC/241010Z/241020Z/B0235E61026/NAVNHQ//

TGTLOC/241015Z/241035Z/B0235E61028//

REFUEL/PIKE 13/6313S/PULLER/ALT:230/240810Z/25/TAD45//

REFUEL/PIKE 16/6316S/PULLER/ALT:230/241099Z/20/TAD43//

MSNDAT/1205H/JAN/SWITCH 05/2FA18/INT/-/BEST/-/20000/32005//
TGTLOC/241020Z/241040Z/B0745CA4579/ALOGHQ//
REFUEL/PIKE 18/6316S/PULLER/ALT:230/241020Z/10/TAD44//

Figure 3-3. Sample ATOCONF Message

3.6.1.2.2 Table Format**.**

When preparing query results in table format, the CMP assumes a relation between the data elements specified in the SELECT clause, and preserves this relation in the output. The query results are printed as a relational table. Each line contains one *tuple*, or combination of the selected data elements. Each tuple element is surrounded by double quotes and separated by commas.

The rows of the table output are formed by taking the columns of data items retrieved for the individual data element specifiers and forming all possible combinations (i.e., one choice from each column of data). Some of these combinations are eliminated by the following rules:

(1) No combinations within a set are allowed. If two data element specifiers are part of the same set in the message format, then the corresponding items in each tuple must be part of the same set in the message text.

(2) No combination may include data items from different segments.

Suppose we again select the MSNDAT.1, TGTLOC.1, and TGTLOC.2 fields from the message in Figure 3-3, and this time produce output in TABLE format. We would start with the following 18 possible output rows (The row numbers, in italics, are not part of the actual output rows.):

row 1	"1201I",	"241010Z",	"241020Z"
row 2	"1201I",	"241010Z",	"241035Z"
row 3	"1201I",	"241010Z",	"241040Z"
row 4	"1201I",	"241015Z",	"241020Z"
row 5	"1201I",	"241015Z",	"241035Z"
row 6	"1201I",	"241015Z",	"241040Z"
row 7	"1201I",	"241020Z",	"241020Z"
row 8	"1201I",	"241020Z",	"241035Z"
row 9	"1201I",	"241020Z",	"241040Z"
row 10	"1205H",	"241010Z",	"241020Z"
row 11	"1205H",	"241010Z",	"241035Z"
row 12	"1205H",	"241010Z",	"241040Z"
row 13	"1205H",	"241015Z",	"241020Z"
row 14	"1205H",	"241015Z",	"241035Z"

row 15	"1205H",	"241015Z",	"241040Z"
row 16	"1205H",	"241020Z",	"241020Z"
row 17	"1205H",	"241020Z",	"241035Z"
row 18	"1205H",	"241020Z",	"241040Z"

Rule 1 eliminates 12 of these rows. For example, row #2 is eliminated because the values of the TGTLOC.1 and TGTLOC.2 fields do not come from the same set in the message text. Rule 2 eliminates another three of the remaining rows. For example, row #9 is eliminated because the MSNDAT.1 data item comes from the first MSNDAT segment, and the TGTLOC data items come from the second segment. After applying the elimination rules, the following three rows remain as the output of the query:

row 1	"1201I",	"241010Z",	"241020Z"
row 5	"1201I",	"241015Z",	"241035Z"
row 18	"1205H",	"241020Z",	"241040Z"

3.6.1.2.3 Errors Format**.**

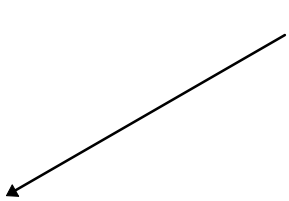
This format is unlike the other two. Instead of arranging data items extracted from the message, it produces a detailed listing of any validation errors detected in the message. This format can only be used when the entire message has been selected. The query used to produce the error report is "SELECT*FORMAT ERRORS."

The error report for a message first lists the segmentation hierarchy of the message. Segments are indicated by indentation. Each new segment at a deeper level of nesting is presented at a new, deeper indentation in the report. Each set is labeled with its presentation number. This section of the report shows missing sets, sets out of sequence, and improperly composed sets (e.g., sets which are missing a mandatory field). Next, the error report shows the message text. This is where invalid fields are reported. The error report for the sample ATOCONF message in Figure 3-4 is shown below. This report shows two validation errors in the message. The second TGTLOC set is missing its fourth, mandatory field. Also, the fifth field in the second REFUEL set is incorrect as it contains a date-time group.

ATOCONF 281456Z APR 93 LOCAL

*** Set Sequence Validation***

(2) OPER
(3) MSGID
(8) PERID
(9) AIRTASK
(10) TASKUNIT
(11) MSNDAT
(13) TGTLOC



(13) TGTLOC

Set Error:

Missing mandatory field

(19) REFUEL

(19) REFUEL
(11) MSNDAT
(13) TGTLOC
(19) REFUEL

Error#2:
The fifth field in this set is incorrect.
This field is FFIRN/FUD 143/109.
("99" is not a valid part of the date-time group)

Field Validation

OPER/JMAPS DEMO//

MSGID/ATOCONF/USCENTAF-CMBT PLANS/040101/APR//

PERID/312320Z/TO:010501Z//

AIRTASK/UNIT TASKING//

TASKUNIT/C41//

MSNDAT/1201I/JAN/EAGLE 01/3A6E/INT/-/BEST/-/20000/32001//

TGTLOC/241010Z/241020Z/B0235E61026/NAVNHQ//

TGTLOC/241015Z/241035Z/B0235E61028//

REFUEL/PIKE 13/6313S/PULLER/ALT:230/240810Z/25/TAD45//

REFUEL/PIKE 16/6316S/PULLER/ALT:230/241099Z/20/TAD43//

Field Errors:

Above alpha or numeric range

Position: 5 FFIRN/FUD: 143/109

MSNDAT/1205H/JAN/SWITCH 05/2FA18/INT/-/BEST/20000/32005//

TGTLOC/241020Z/2411040Z/B0745CA4579/ALOGHQ//

REFUEL/PIKE 16/6316S/PULLER/ALT:230/241020Z/10/TAD43//

Figure 3-4. An Error Report for an ATOCONF Message

The value "99" in the "minutes" component of this field is not within the allowable range of values.

3.6.1.2.4 Report Format**.**

Many people do not know the meanings of the fields in a message or the coded values of these fields. The report format produces an easy to understand version of a message by annotating each

field with text describing its format and use. The result is an expanded version of the message text in which all the message information is printed in plain text for the non-expert reader.

When the processor prints a message in report format, it prints the contents of each set plus additional text describing the contents of that set. Linear and columnar sets receive different treatment during this process. For example, a linear set extracted from a TACELINT message in report format might appear as follows:

PRM

DATA ENTRY::	01
RADIO FREQUENCY::	00895.5MHZ
RF OPERATIONAL MODE::	D(DISCRETE FREQUENCY)
PULSE REPETITION INTERVAL IN MICROSECONDS::	001085.897
PRI ACTIVITY CODE::	S(PULSE STAGGER)
PULSE DURATION IN MICROSECONDS::	0.540
SCAN TYPE::	STDY (STEADY)
SCAN RATE::	-

According to the standard, the third field in a PRM set represents the "RF [radio frequency] operation mode." The code "D" indicates that the operational mode is "discrete frequency." Both facts are clearly reflected in the report format output.

Because the fields in a columnar set may be repeated many times, it is not practical to annotate each field with descriptive text. Instead, the field contents are preceded by a description of each column in the set. The column description applies to each element in the column. Figure 3-5 shows a columnar set extracted in report format from an AIRSUPREQ message:

In this example, the seventh column (WPNTY) contains two alternate field types. The descriptive text for this column includes the description of both field types.

GENTEXT and free-text sets are exempted from report formats. The descriptive text associated with the fields of these sets does not add to the reader's comprehension, and is therefore suppressed.

3.6.1.3 The FROM Clause.

There may be at most one FROM clause in every query. The FROM clause describes the messages from which data elements will be extracted. It is primarily useful in queries that are intended to be applied to all the messages in the message repository. (It may be omitted from queries used only in routing-table entries; these queries are always applied to a single message.) The FROM clause describes the message domain in terms of one or more of the following three features:

a. the message **type name** (MSGID)

b. the message **originator** (ORIG)

c. the time of transmission, known as the message **date-time group** (DTG).

A MSGID term compares the message type name to an alphanumeric string. It may specify all messages of a particular type. It may also specify all messages that are not of a particular type. For example:

```
FROM MSGID="LOCATOR"      -to select all LOCATOR messages
FROM MSGID!="LOCATOR"     -to select everything except LOCATOR
```

An ORIG term compares the message originator to an alphanumeric string. For example:

FROM ORIG="HQ ACC LANGLEY AFB VA//DRIS//"

```
FROM ORIG!="HQ ACC LANGELY ABF VA//DRIS//"
```

```
8FACSCD
   /REF   /ATKACCS      /MSNNO   /ATIME    /CAA/ACTYP  /WPNTY   /CMNT
   /CA0011/BRFKO SWILL  /CAS011  /231345Z/   18/A3     /AGM69A          original
   /CA0012/SANDY        /CAS012  /231445Z/    4/A3     /AGM69A          columnar
   /RE0014/TOP GUN      /CAS015  /231515Z/    4/F111A  /ATOG            set
   /CF16A /HOT SHOT     /CAS017  /232015Z/    3/A10A   /AGM65B//

8FACSCD
     COL  HEADING      DESCRIPTION                    columnar set in
      01  REF          REFERENCE NUMBER               report format
      02  ATKACCS      ATTACK AIRCRAFT CALL SIGN
      03  MSNNO        MISSION NUMBER
      04  ATIME        AIRCRAFT ARRIVAL DAY-TIME
      05  CAA          COUNT OF AIRCRAFT ALLOTTED
      06  ACTYP        AIRCRAFT TYPE/MODEL
      07  WPNTY        ORDNANCE LOAD CODE, or
                       WEAPON TYPE, AIR-TO-SURFACE

 REF    ATKACCS      MSNNO    ATIME     CAA   ACTYP  WPNTY   CMNT
 CA0011 BRFKO SWILL  CAS011   231345Z    18    A3     AGM69A
 CA0012 SANDY        CAS012   231445Z     4    A3     AGM69A
 RE0014 TOP GUN      CAS015   231515Z     4    F111A  ATOG
 CF16A  HOT SHOT     CAS017   232015Z     3    A10A   AGM65B
```

Figure 3-5. Columnar Set in Report Format

The first clause chooses all messages sent by the DRIS office of Air Combat Command Headquarters at Langley Air Force Base (AFB). The second clause chooses all other messages.

The alphanumeric strings in MSGID and ORIG terms may contain any sequence of characters. However, these strings must be an exact match for the message text. For example, the clause:

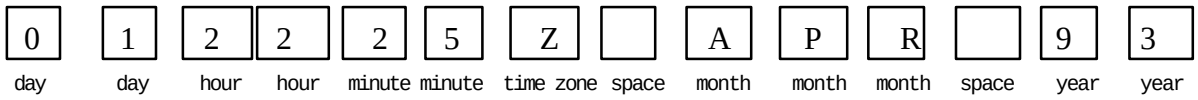
FROM MSGID="LOCATE"

is legal, but will never select any messages, since "LOCATE" is not a valid MTF message type name. Likewise, the clause:

FROM ORIG="HQ ACC LANGLEY A.F.B. VA//DRIS//"

will not select any messages from "HQ TAC LANGLEY AFB VA//DRIS//" because the originator strings are not an exact match. Consequently, care must be exercised in composing FROM clauses that contain MSGID or ORIG terms.

A DTG term compares the message transmission time to a string representing a date-time group. A date-time group string contains exactly 14 characters in the following format:



Any of the following comparison operators may be used:

=	equal to	(message sent at this time)
!=	not equal to	(message not sent at this time)
>	greater than	(message sent after this time)
>=	greater than or equal to	(message sent at this time, or later)
<	less than	(message sent before this time)
<=	less than or equal to	(message sent at this time, or earlier)

For example, this clause selects all messages sent before 15 June 1991:

FROM DTG<"160000Z JUN 91"

A FROM clause may contain a combination of MSGID, ORIG, and DTG terms. These terms are combined with the Boolean operators AND and OR. The AND operator has precedence over the OR operator. Parentheses may be used to override this default precedence. For example, the following clause selects all TACELINT and LOCATOR messages sent before 15 June 1991:

FROM (MSGID="TACELINT" OR MSGID="LOCATOR")

AND DTG<"160000Z JUN 91"

A wildcard may be used in a MSGID, ORIG, or DTG term. For example, this clause selects all messages of any type:

FROM MSGID="*"

3.6.1.4 The WHERE Clause.

A query may contain at most one WHERE clause. The WHERE clause places restrictions on the data to be extracted. It contains one or more conditional terms that must be satisfied by the contents of each message from which data elements are to be extracted. These terms are combined using AND and OR operators, using the precedence rules described for FROM clauses.

Terms in a WHERE clause have one of the following forms:

a. They may test whether a data element exists in the message. For example, to specify that the message must contain an ATTACK segment, we use the term "[ATTACK] EXIST." To specify that the message must not contain a TMPOS set, we write "TMPOS !EXIST."

b. They may test whether a specific field's contents were formatted using a given FFIRN/FUDN. These are valid terms of this type:

ATTACK.1 =32/2;

TMPOS.2! = 323/2

c. They may test whether a specific field matches a string constant. These are valid terms of this type:

ATTACK.1="MK48"

TMPOS.3!="180T"

d. They may compare a specific numeric field with a numeric constant. These are valid terms of this type:

SUB.1>2

NAVAL.1<=3

e. They may compare two numeric or string fields. These are valid terms of this type:

SUB.1>SUB.2

[SUB]ATTACK.1!=[NAVAL]ATTACK.1

The WHERE clause, if included, must follow the SELECT clause. Every field referenced in the WHERE clause must be included in the SELECT clause.

The effect of the WHERE clause is to eliminate lines of output that would otherwise be produced by the query. Conceptually, the test in the WHERE clause is applied to every line of output. If the test fails, the output line is not printed. A WHERE clause is most meaningful when a query either selects a single data element or uses the table output format. It has no meaning when the errors format is used.

3.6.1.5 The WITHIN Clause.

A query may contain several WITHIN clauses. A WITHIN clause restricts data extraction to a specified message segment. For example, if we are only interested in extracting information from SUB segments, we can write:

WITHIN [SUB]

SELECT SUB.1, TMPOS.1, TMPOS.2

The WITHIN clause simply serves as a shorthand for the following query:

SELECT [SUB]SUB.1,[SUB]TMPOS.1,[SUB]TMPOS.2

The restriction of a WITHIN clause applies only to subsequent parts of the query; it does not apply to any preceding clauses. In the following example, the WITHIN clause restricts only the WHERE clause:

SELECT TMPOS.1

FROM MSGID="*"

WITHIN [NAVAL]

WHERE BRNG EXIST

The query extracts the first field of every TMPOS set in every message where there is a NAVAL segment containing a BRNG set. These fields will be extracted regardless of whether the TMPOS set is located within a NAVAL segment. To accomplish this latter restriction, we would use the following query:

WITHIN [NAVAL]

SELECT TMPOS.1

FROM MSGID="*"

WHERE BRNG EXIST

A query can contain more than one WITHIN clause. The restriction imposed by the first clause extends only to the start of the second clause. For example, the following query extracts every

TMPOS set contained in a NAVAL segment, but only from those messages where a BRNG set exists within a SUB segment:

WITHIN [NAVAL]

SELECT TMPOS
FROM MSGID="*"
WITHIN [SUB]
WHERE BRNG EXIST

The query could be rewritten without WITHIN clauses as:

SELECT [NAVAL]TMPOS

FROM MSGID="*"
WHERE [SUB]BRNG EXIST

In general, WITHIN clauses are simply a shorthand notation used instead of prefixing every set and field specifier with the segment specifier. This will help you to write simple, concise queries. However, there are two special cases where a WITHIN clause does not simply take the place of a segment prefix:

a. The clause "WITHIN[*]" restricts selection to data elements that appear within any segment. Elements that are not part of any segment are excluded.
b. The clause "WITHIN []" eliminates the restriction imposed by a preceding WITHIN clause.

3.6.1.6 The VALIDATE Clause.

The VALIDATE clause controls the parts of the message that are validated during the execution of the query. Any errors in these parts of the message will be reported. Other errors will be ignored.

There are two reasons for providing validation controls in a query. The first reason is that client applications differ in their sensitivity to validation errors: some care about errors anywhere in a message, some care about errors in the data elements they require, and some do not care about errors at all. Validation controls make it possible to suppress the error reports that are not wanted by the client. The second reason is that by avoiding unnecessary validation work, the CMP can run faster.

There are four possible values for the VALIDATE clause:

(1) NONE: no validation is performed and no errors will be reported. This option is used for maximum performance whenever error detection is not important.

(2) EXTRACTED: the contents of the data elements selected by the query are validated. The set sequence of the entire message is also validated. No structural validation is performed. Errors in the selected data elements in the message are reported; errors elsewhere in the message are ignored and are not reported. This option is used to avoid the expense of validating the entire message in those cases where only errors in the extracted information must be detected.

(3) MESSAGE: the entire message is validated. All types of validation are performed, and errors anywhere in the message are reported. No special testing is done to the selected elements. This option is used in those cases where any error anywhere in a message makes the whole message untrustworthy.

(4) ALL: a combination of the previous two options. The error report will indicate one of three conditions:

(a) No errors in the message

(b) Errors in the selected data elements.

The VALIDATE clause is optional. It may appear at most once in a query. The default value is NONE, performing no validation.

The processor reports the results of the specified validation on the .ERRORS line in the output protocol, which is fully described in Section 3.8.2. The validation result is presented as an integer.

This number must be interpreted as a collection of independent values represented by specific bit positions. These values are represented as follows:

a. Bit 0 (the least-significant bit) is set if the parser found errors anywhere in the message. Suppose the validation result is contained in the variable *errs*. Then, a client program written in C can test this flag with the expression *errs & 01*.

b. Bit 1 is set if errors were found in the selected data elements. Client programs test this flag with the expression *errs & 02*.

All other bits are unused at present. However, client programs must not depend on these bits containing zero because they may be used in a later release.

It is important to realize that the validation results detailed above depend on both the errors present in the message and the setting of the VALIDATE clause in the query. For example, the fact that errs & 01 is zero means either that the message is completely valid, or that the parser was not asked to validate the message. Table 3-1 shows the validation result returned for all possible combinations of VALIDATE levels and message errors.

Table 3-1. Relation of Validation Levels and Results

VALIDATION LEVEL	NO ERRORS	ERRORS ONLY IN SELECTED FIELDS	ERRORS ONLY IN NON-SELECTED FIELDS	ERRORS IN BOTH SELECTED AND NON-SELECTED FIELDS
NONE	0	0	0	0
EXTRACTED	0	3	0	3
MESSAGE	0	1	1	1
ALL	0	3	1	3

3.6.2 Shells

The purpose of a shell is to take the output from one or more ordinary queries and to substitute the data items extracted by these queries into a text template. This provides the capability to produce pure SQL statements for updating a database. It also provides a simple report generator capability. For example, in the section describing the FORMAT clause (Section 3.6.1.2), the query produced output in tabular format, appearing as follows:

"1201I", "241010Z", "241030Z",
"1205H", "241020Z", "241040Z".

By placing the same query inside a shell, we can produce the following SQL statements:

UPDATE MISSION_TIMES
SET MISSION_START_TIME="241010Z",

MISSION_END_TIME="241030Z"
WHERE MISSION_ID="1201I"

SET MISSION_START_TIME="241020Z",

MISSION_END_TIME="241040Z"

WHERE MISSION_ID="1205H"

With a different shell, we could produce the following report:

Mission 1201I begins at 241010Z, ends at 241030Z.
Mission 1205H begins at 241020Z, ends at 241040Z.

A shell consists of an optional syntax declaration section and one or more scope sections. The syntax declarations define the characters used to begin and end a scope section. Each scope section produces one block of text in the final output. A scope may contain a query, a text

template, both, or neither, for example, the shell used to produce the SQL statements shown above contains two scopes: one containing only text, and one containing text and a query.

3.6.2.1 Syntax Declaration.

Shells can contain the C-style comments or C++-style comments, which start with a "//" token and continue through the next newline character.

By default, the left and right brace characters, '{' and '}' respectively, mark the beginning and end of each scope. However, the syntax declaration section can be used to define alternative characters for this purpose. Each must be a single character, and neither can appear in any other context within the shell, except within comments. When present, the syntax declaration must appear before the shell's first scope. For example, the following statements declare that square braces are to be used to delimit scopes:

a. #define start [
b. #define end]

3.6.2.2 Scope.

A scope contains a query part and/or a text part. The query part of the scope begins with the token and ends at the first subsequent semicolon. It must contain a valid query. Any other text comprises the text part, or template, of the scope. The template can contain any kind of textual information, including placeholders.

A placeholder is an ampersand character ('&') followed by an integer; it shows where substitution of retrieved data into a copy of the template will occur. The substitution process is repeated for each row of data produced by executing the query. For example, the placeholder "&2" represents where to substitute data items extracted by the second data element specifier in the scope's query SELECT clause.

It should be noted that if a template contains placeholders, a query part must appear within its scope. Furthermore, there must be at least as many elements in the query's SELECT clause as the highest-numbered placeholder in the template; that is, there cannot be a placeholder "&5" unless there are at least five elements in the query's SELECT clause.

When both are present in a scope, either the text or the query part may appear first. The scope's query part may even appear between two text parts, although during execution the two text parts will be treated as though they had appeared contiguously. The query must use the FORMAT TABLE output clause; any other FORMAT is flagged as an error.

3.6.2.3 Shell Output.

The output of a shell is comprised of its separate scopes' outputs. The output of each scope depends on whether it contains a query part, a text part, both, or neither. Table 3-2 summarizes the possible scope outputs for valid combinations of queries, templates, and placeholders. For example, if the scope contains only a query, any legal output format may be specified and the output of the scope is the output of the query. On the other hand, if the scope contains only a template (with no placeholders), then a single copy of that text comprises the scope's output. Also, if the scope contains both a template and a query, but the query produces no output, then neither does the scope.

Table 3-2. Summary of Scope Output

Query	FORMAT	Template	Placeholders	Scope Output
No	N/A	No	N/A	None
No	N/A	Yes	Not allowed	One copy of text in template
Yes	Any	No	N/A	Results of executed query
Yes	Any	Yes	No	One copy of text in template and results of executed query
Yes	TABLE	Yes	Yes	*n* copies of text in template with query result substitutions, where *n* is the number of rows in the query result

Although seldom of importance in queries, space and newline characters are significant in the text part of a scope, as are upper- and lower-case characters. All are reproduced exactly in the shell's output. Table 3-3 shows some examples of how newline characters affect the output of a scope.

Table 3-3. Effect of Newline Characters in Scope

QUERY	QUERY OUTPUT
SELECT ITEM.1, ITEM.2 FORMAT TABLE	"123", "456" "789", "OAB"

scope	**scope output**
{SQL:	A=123, B=456,;A=789,B=0AB;
SELECT ITEM.1,ITEM.2	
Format Table;	
A=&1, B=&2;}	
{SQL: SELECT ITEM.1,ITEM.2 FORMAT TABLE;	A=123, B=456 A=789, B=0AB
A=&1, B=&2;	
}	
{SQL: SELECT ITEM.1, ITEM.2 FORMAT TABLE; *[blank line]* A=&1, B=&2;	*[blank line]* A=123, B=456 *[blank line]* A=789, B=0AB
}	

query **query output**

3.6.2.4 Output Directive.

By default, the shell outputs from scopes are concatenated together and sent to the display screen or to an awaiting application. But it is also possible to use a special shell directive to indicate the line printer, the screen, or a named file as the default destination for scope output. The following are examples of valid shell output directives:

a. #output PRINTER //default output goes to the line printer
b. #output SCREEN //default output goes to the screen

c. #output "Shell.Out" //default output goes to the named file.

Where output directives appear in the shell determines which of its scopes are affected. Multiple directives can be provided, but each one must be positioned outside of any scope (e.g., before the first scope or between one end of scope marker and a subsequent beginning of scope marker). When a directive is present, the output from any scope appearing after it in the shell but before

another output directive will be written to the specified destination. Any scope that is not preceded by a shell output directive will write its results to the normal default location.

For example, suppose a shell contains two scopes and you want both scopes' outputs to be written to a file called "SUMMARY.dat". You could write as first line of the shell the output directive naming "SUMMARY.dat" to accomplish this.

If the execution of the scope's query does not produce any output rows, then there is nothing to substitute into the template. In this case, the scope does not produce any output.

If a scope has both a query part and a text part, but the text part of a scope does not contain any placeholder tokens, then no text substitution is performed. Instead, the query is executed, and the scope's output then consists of the results of the query and a copy of the text part.

3.7 USER INTERFACE REFERENCE

The UI is a window-based application using the X Window System and the Motif Graphical UI toolkit. The UI allows users to tell the processor about new client applications, ask questions about the messages that have arrived, and make corrections to messages that contain errors. This section describes how to start the UI, and manipulate the UI with the keyboard and a mouse, and lists the display windows and functions available within the UI.

3.7.1 Starting the User Interface

Before users can start the UI, the administrator must have already started the message processor. Also, users must belong to the **CMP** group, and must have the parser commands directory (usually/**usr**/**local**/**parser**/**bin**) in their PATH variable.

The X Window system and the Motif window manager must be running on the user's terminal before the UI can be started. On some systems, X and Motif will be started automatically (by the **xdm** program) when the user logs in. On other systems it is necessary for the user to start X manually, using the **startx** or **xinit** commands. Some SUN systems use Open Windows, which is a superset of X, and on these the user must use the **openwin** command.

To start the user interface, enter **parser ui** at the UNIX prompt. After a brief delay, the UNIX prompt will reappear. After a slightly longer delay, the UI window will appear on the display.

3.7.2 Controlling the User Interface

3.7.2.1 Mouse Buttons.

The interface is based on the Open Software Foundation (OSF)/Motif user environment, which is in turn based on the X Window System. In this environment the processor displays its output in "windows" on the display, and the user supplies new commands with a mouse in addition to the keyboard. The following terminology is widely used to describe the use of the mouse and its buttons:

a. To *point* to something on the display, move the mouse until the pointer is positioned over the item.
b. To *click* on something, point to it, then press and release the mouse button without moving the mouse. Generally you will use the *select* button, which is the left button on the mouse.
c. To *drag* something, point to it, then move the mouse while pressing and holding the mouse button.

3.7.2.2 Point and Click Commands.

There is a set of "point-and-click" commands that use the mouse and are common to all Motif applications. These include:

a. moving a window (by dragging its title bar)
b. re-sizing a window (by dragging the window's frame)
c. changing a window into an icon (by clicking the minimize button)
d. choosing from a pull-down menu (by clicking on the menu bar).

There may be several windows on the display. Only one window, the active window, receives input from the keyboard. To activate a different window, point to the window and click the select button. The window frame of the selected window will change color to remind you that it is now the active window.

Many operations are chosen through the pull-down menu. When prompted in the manual to "choose Windows Queries," first point to "Windows" in the menu bar, click the select button, and then point and click on "Queries" in the menu that appears. The UI uses several standard Motif controls for user input, including check boxes, input fields, and push buttons.

A check box is a toggle switch that turns some feature on and off. The feature is "on" when the box is dark and depressed; it is "off" when the box is light and raised. To toggle a check box, point to it with the mouse and click the select button.

An input field is where text is entered to be read by the CMP. In this manual, when instructed to "type into an input field," first click it with the mouse (to select it) and then enter text using the

keyboard. The selected (or "active") input field has a dark solid border and contains a blinking text cursor. The backspace, delete, and cursor arrow keys can all be used to edit the entered text.

3.7.2.3 Push Buttons.

A push button performs some immediate action. In this manual, when instructed to "press" a push button, point to it with the mouse and click the select button. Certain push buttons perform the same action in several different windows. These are the Update, Close, OK, and Cancel buttons.

a. UPDATE - When this button is pressed, the contents of the active window are regenerated. This is necessary because the processing requirements do not allow these windows to be updated in real time. The system may receive data or another user may change the state of the system; these events will not be reflected in the contents of the current window until the user presses the Update button.
b. CLOSE - The Close button will close the current window.
c. OK - The OK button will cause the currently requested operation to take effect, using the information entered into any input fields and check boxes. The window is then closed.
d. CANCEL - The Cancel button will cancel the current operation. Any information entered into input fields or check boxes is discarded.

The UI uses certain standard Motif controls to display information and to permit the user to make selections from the display. These include list boxes and scroll bars. Each line in the list box displays a single list item. To select a list item, point to it and click the select button; the selected item will then be displayed in reverse video. Some of the commands available through the pull-down menus apply to the selected list item. For example, if the Command Delete were chosen, then the "locator" list item would be deleted.

A scroll bar appears whenever there are too many items to fit into a list box. You can control the portion of the list that is displayed in the list box by clicking on either of the arrow buttons, or by dragging the slider up or down to a new position in the scroll bar.

3.7.3 Functions Available through the User Interface

Most of the functions required to interact with the parser are represented as options on the *Windows* pull down menu pane. Each option invokes a new window specific to its associated function. For example, selecting *Routing* from the *Windows* pull down menu will activate a window allowing access to the parser routing database.

Once a window has been activated and displayed, there are typically two ways to access the operations available from within that window:

a. The *Commands* pull down menu: This menu contains actions common to most of the windows. For example, the *Message Repository*, *Queries*, and *Log* windows all make use of *Delete* and *Print* on the *Commands* pull down menu.

b. Push button within each window: A control region containing push buttons exists in each window and offers actions specific to a particular window. For example, the *Queries* window provides an *Execute* button that submits a selected query to the parser. This action is specific to *Queries* and thus does not appear elsewhere.

3.7.3.1 The Message Journal Window.

The Message Journal is a data storage area for all messages retained by the processor. Access to stored messages is obtained through the Message Repository window, which provides functions allowing a user to maintain and browse the message database. Each of these functions is described in detail in the following sections.

3.7.3.1.1 View Messages.

To view the text of a message saved in the message repository, select an item from the displayed list, then either choose *Commands View* from the master menu or press the *View* button in the window's control region. The text of the message will appear in a newly created window on the screen.

3.7.3.1.2 List Messages.

The List command allows you to list messages selectively in the message repository. By default, the message repository lists all the messages that have been saved. To use the List command, activate the *List* button. List will prompt for a message identifier, date-time group, Plain Language Address (From, To) and originator, and will then list only those messages that match these input specifications. For example, if you specify ATOCONF for the message identifier, then only ATOCONF messages will be listed in the message repository window. Note that the list criteria will remain in effect until modified again.

3.7.3.1.3 Delete Messages.

The Delete commands (found only on the *Commands* pull down menu pane) allow you to delete messages from the message repository.

To delete a single message, select the message in the message repository, then choose *Commands Delete*.

To delete a group of messages, choose *Commands Delete By*. This will produce the Delete By window. Delete By will prompt for a message identifier, date-time group, Plain Language Address (From, To) and originator, and will then delete only those messages that match these input specifications. For example, if **ATOCONF** is entered for the message identifier, then all the ATOCONF messages in the message repository will be deleted.

3.7.3.1.4 Correct Messages**.**

If errors are discovered in a message that has been saved in the message journal, then corrections can be made via the message editor. To use the processor to correct a message in the message journal, select the message to be corrected and press the *Correct* button at the bottom of the Message Journal window. The window will then appear and corrections can be made. Refer to the User's Manual for instructions on CMP operation. When all the corrections have been made, exit the processor, and the corrected message will be returned to the message journal.

3.7.3.1.5 Submit Message**. ((Not currently implemented))**

The *Submit* command takes a message from the message journal and causes it to be processed in the manner specified by the routing information for that message type. In essence, *Submit* simulates the actual receipt of the message from the communications port. This can be useful when errors are detected in messages. The objective CMP will route messages containing errors to an interactive message queue. Currently, when an error is found, the processor can be used to correct the message. Once corrected, the message will appear in the Message Repository. *Submit* is then used to feed the message through the processor, extract the required data and route it to any waiting application. *Submit* can also be useful for programmers and administrators when setting up new or modifying existing application and routing information. When a message, is submitted, the routing, application, and query information are executed for this message as if it were a newly received message. Any existing query will be applied to the message, and the results will be routed to the application specified in the routing information.

To submit a message for further processing, select the message in the Message Journal window and activate the *Submit* button.

3.7.3.1.6 Update**.**

The Message Journal window's message list is not automatically updated as new messages arrive in the database. To display any newly arrived messages, activate the *Update* button in the window's control region.

3.7.3.1.7 Report**.**

This command generates an annotated listing of a selected message in a separate display window. To select this command, press the *Report* button in the control region.

3.7.3.1.8 Errors**.**

This command generates an error report for a message describing the results of the validation process. This report can be displayed for any message in the database by selecting the message and pressing the *Errors* button in the control region.

3.7.3.2 The Queries Window.

Queries are used to specify data to be extracted from messages. The Query Repository is a data storage area for all queries known to the processor. This database can be accessed by selecting *Queries* from the *Windows* pull down menu. This will produce a list of available queries in the Queries window. Actions associated with the Query Repository are described in detail in the following sections.

3.7.3.2.1 New**.**

To create a new query, choose *Commands New.* This will produce a window that contains a box for the query name and a box for the text of the query. Enter the name and the query text, and then click the Validate button. If the query is valid, press the *OK* button and the query will be added to the query repository. Pressing the *OK* button will also validate the query and alert the user if it is invalid. The system will not add an invalid query to the Query Repository.

3.7.3.2.2 Edit**.**

To change the text of a query, select the name of the query to be modified and choose *Commands Edit*. This will produce a window containing the query name and its text. After modifying the text, press the *Validate* button to check the validity, or *OK* to commit the change to the Query Repository. The system will not allow an invalid query to enter the Query Repository.

3.7.3.2.3 View**.**

To view a query, select the name of the query in the Queries window and choose *Commands View.*

3.7.3.2.4 Delete**.**

To delete a query, select the name of the query in the Queries window and choose *Commands Delete*.

3.7.3.2.5 Save**.**

To save a query under a new name, select the name of the query in the Queries window and choose *Commands Save As*. A window will appear containing the text of the selected query and a blank field for the new name. Fill in the name of the new query and press the *OK* button.

3.7.3.2.6 Print**.**

To print a query, select the name of the query in the Queries window and choose *Commands Print*.

3.7.3.2.7 Execute**.**

To execute a query interactively, select the name of the query in the Queries window and press the *Execute* button. The results of the query will be displayed in a new window and also saved in the Query Report Repository. The report can be identified in the repository by the time that it was created.

3.7.3.2.8 Update**.**

The Query Repository window's query list is not automatically updated as new queries arrive in the database. To display any newly entered queries, press the *Update* button in the window's control region.

3.7.3.3 The Query Reports Window.

A query report contains the results of the interactive execution of a query on a message. See Section 3.5.2.2 for information on interactive query execution. To access query reports, choose *Windows Query Reports*. Available actions are described in the following sections.

3.7.3.3.1 View**.**

To view a query report, select the name of the report in the Query Report window and choose *Commands View.*

3.7.3.3.2 Delete**.**

To delete a query report, select the name of the report in the Query Report window and choose *Commands Delete*.

3.7.3.4 The Routing Window.

Routing is used to tie queries and applications together. The query will run against a message received by the CMP, causing a query report to be generated. This report is then sent to the associated application specified in the routing table.

The routing window lists the message types that are of interest. The "SAVE" flag indicates that the particular message type is saved after being received by the system. Available actions are described in the following sections.

3.7.3.4.1 New.

To add a new message type to the routing tables, choose *Commands New*. This will produce a window containing a text box for the message type and a check box for SAVE. Enter the message type in the text box and click the check boxes for appropriate values. Then press the *OK* button.

3.7.3.4.2 Edit.

To change the flag for a particular message type, select the message type to be modified and choose *Commands Edit*. This will produce a window containing the message type name and the check box for SAVE. Modify the flag as desired and press the *OK* button.

3.7.3.4.3 Delete.

To delete routing for a message type, select the message type in the Routing window and choose *Commands Delete.*

3.7.3.4.4 Save.

To save one message type as another message type, select the message type to be replicated and choose *Commands Save As*. This will produce a window with a text box. Fill in the text box with the new message type and press the *OK* button.

3.7.3.4.5 Applications.

To access applications related to a message type, select the message type of interest in the Routing window and press the *Applications* button in the control region. This will produce a

window that lists the applications associated with the selected message type. Actions available in the Applications window are described next.

3.7.3.4.6 New**.**

To add a new application, choose *Commands New*. This will produce a window with text input fields and check boxes. The text fields are for application name, the host where the application resides, and the name of the query that will produce a report for the application.

Fill in the text for application, host, and query name, click the appropriate validation boxes, and press the *OK* button. Note that the query named must already exist in the query repository before the new application is added to the routing tables.

3.7.3.4.7 Edit**.**

To edit an application, select the application in the Applications window and choose *Commands Edit*. This will produce a window with the application information. Make the necessary changes and press the *OK* button.

3.7.3.4.8 View**.**

To view an application, select the name of the application in the Applications window and choose *Commands View.*

3.7.3.4.9 Delete**.**

To delete an application, select the name of the application in the Applications window and choose *Commands Delete*.

3.7.3.4.10 Save**.**

To save an application under a new name, select the name of the application in the Applications window and choose *Commands Save As*. This will produce a window with the application information. Fill in the application text field and press the *OK* button.

3.7.3.5 The Log Window.

The audit log provides a sequential list of major events that occur within the processor. Event descriptions are provided along with the date and time that the event occurred. Examples of the types of events reported to the Log are: creation of processes during startup, or receipt of a new message. This serves to provide the user or system administrator with an audit trail of important

transactions and events. The Log may be cleared at any time by choosing *Commands Delete* when the Log is the active window.

3.7.3.6 The Message Error Log Window.

This feature is presently not implemented. In future releases the processor will display message errors in this window as they are discovered during message parsing or query execution. This window will be scrollable, providing a way to review all errors found in messages.

3.8 THE APPLICATION PROGRAM INTERFACE

Client systems use the processor to validate incoming messages and extract the information of interest. This section describes the interface between the processor and these client systems. The first part describes the syntax of *client configuration files*, which specify the services required by a client. The second part describes the *output protocol*, which specifies how information is sent from the CMP parser to client programs. The last part describes the *input protocol*, which specifies how a client program can submit a message to the CMP for processing. (For more in-depth information concerning API configuration see the CMP API definition document.)

3.8.1 Client Configuration Files

A client configuration file contains the information required to register a client program with the message processor. This information specifies the types of messages of interest, the data items to be extracted from these messages, and the destination to which the extracted data should be routed. Client configuration files are ordinary text files and can be edited with any text editor. The intention is that the developers of a system will compose a client configuration file and distribute this file together with their system software. When this system software is installed at an operational site, the client configuration file will be installed in the configuration directory at the site.

a. The CMP reads the client configuration files at startup time. The processing proceeds as follows:

   (1) Configuration file processing is only performed during a **coldstart** operation. The configuration files will be ignored upon a **warmstart**.

   (2) The configuration directory must be specified by the **ATT.confdir** variable in the **.MAPSconfig** file. If this variable does not exist, then the CMP will not read any configuration files. If the specified directory does not exist or cannot be read, then it terminates with an error message.

   (3) Each file in the directory with a name ending in **.ccf** is read and processed. For example, **demo.ccf** will be processed, while **demo** will be ignored. The files are

processed in alphabetical order. If any of the **.ccf** files cannot be read, the processor terminates with an error message.

(4) Each definition must be completely contained within a single file. It is possible for a definition in one file to refer to a definition in a previously-processed file; however, this is strongly discouraged. As a matter of programming style, each file should be completely self-contained.

b. Each client configuration file contains a series of definitions. Four kinds of definitions may be included in the file:

(1) Query: Defines a query
(2) Shell: Defines a shell, which contains zero or more queries
(3) Routing: Adds a single entry to the routing table
(4) Message: Sets the *save message* and/or *full parse* flags for one message type.

In general, names in a client configuration file are case-insensitive; that is, the parser considers the strings **query**, **QUERY**, and **QuErY** to be identical. Lowercase characters are mapped to uppercase. The only exception to this rule is the command strings that are part of routing definitions (see Section 3.8.1.3).

Comments may be inserted at any point. The '#' character is the comment character. This character and all subsequent characters on the line are ignored.

Every definition in a client configuration file has a name. These names may contain any mixture of alphanumeric characters, hyphens (-), underscores (_), and period (**.**). Furthermore, any string may be used as a name if it is enclosed in double-quote characters. This mechanism permits names that contain spaces; for example, "OPTASK LINK."

3.8.1.1 Query Definitions.

The syntax of a query definition is as follows:

query *name = query-string*

The *name* is the query name submitted to the processor. If another query or shell having the same name has been previously defined, it terminates with an error message.

The *query-string* is the text of the query being defined. It must be a valid query; otherwise, CMP terminates with an error message. Every query contains exactly one semicolon, which must be the last character. This final semicolon also marks the end of the query definition.

3.8.1.2 Shell Definitions.

The syntax of a shell definition is as follows:

shell *name = { shell-string };*

The name is the shell name submitted to the processor. If another query or shell having the same name has been previously defined, it terminates with an error message.

The *shell-string* is the text of the shell being defined. It consists of any sequence of characters in which the number of open-brace ({) characters is equal to the number of close-brace (}) characters. It must form a valid shell; otherwise, the processor terminates with an error message. Unlike a query definition, the final semicolon in a shell definition does not become part of the shell.

3.8.1.3 Routing Definitions.

The syntax of a routing definition is as follows:

route msgtype **query** q-name [**host** hostname] **cmd** cmd-string;

The *msgtype* is the name of the message type being routed.

The **query** section is mandatory. It specifies the name of the query or shell that will be run on each incoming message of the specified message type.

The **host** section is optional. The *hostname* specifies the name of the machine that will run the client application receiving the query results. If this section is omitted, the processor assumes **host localhost**.

The **cmd** section is mandatory. The *cmd-string* specifies the name of the client application that will be executed each time a msgtype message arrives. It consists of any sequence of characters, not including a semicolon (";"). In this string, the case of alphabetic characters is significant; for example, **/usr/local/isum/isum** may not have the same effect as **usr/local/isum/ISUM.**

The command string will eventually be executed by the Bourne shell *sh(1)*, so it is unwise to write a command string that includes characters having special significance to the shell.

There may be several routing definitions for the same message type. The processor will execute the behavior specified for each routing definition whenever it receives a message of that type. The order of execution of the routing behaviors is undefined.

3.8.1.4 Message Definitions.

The syntax of a message-processing definition is as follows:

set *msgtype* [**save** = *saveflag*] [**fullparse** = *parseflag*];

The *msgtype* specifies the type of message.

The **save** section is optional. The *saveflag* specifies whether the parser will retain incoming messages of this type in the message repository. It must be either **yes** or **no**. If this section is omitted, the parser assumes **save=no**.

The **fullparse** section is optional. The *parseflag* specifies whether the parser will completely process each incoming message of this type, or only those portions requested by client applications. It must be either **yes** or **no**. If this section is omitted, it assumes **fullparse=no**.

It is possible to have more than one message processing definition for a message type. When this occurs, the parser assumes **save=yes** if any definition sets **save=yes**, and assumes **fullparse**=**yes** if any definition sets **fullparse=yes**.

The default behavior for message types that do not have message processing definitions may be set by a definition for the **default** message type.

3.8.2 The Output Protocol

Client applications register themselves with the processor by specifying the types of messages of interest, the data items to be extracted from these messages, and the destination to which the extracted data should be routed. When these messages arrive, it performs the specified processing and sends the resulting data to the client applications through the *output protocol*. This protocol includes both the mechanism and the format of the data transfer.

3.8.2.1 The Data Routing Mechanism.

Routing entries specify a message type, a query, a host name, and a command string. When a message of the specified type arrives, the processor applies the query to the message, saving the output in a temporary file. It then executes the command string on the specified host using the UNIX *system(3)* function and the *rsh(1)* command. For example, given a new TACELINT message and the following routing definition:

route tacelint query isum-01 host bartender cmd/usr/isum/isum;

the processor first saves the output from *isum-01* into a temporary file, and then executes the following string with the *system(3)* function:

rsh bartender/usr/isum/isum<*tempfile* &

The effect is to redirect the query output to the standard input of the *isum* command, which is executed on the remote host bar*tender*.

Here are some observations about the routing mechanism:

a. If the specified host is the special string *localhost*, then the parser omits the "rsh *hostname*" from the command string sent to *system(3)*. The command is instead executed on the machine executing the message processor.

b. The executed command is not expected to produce any results on either its standard output or standard error stream. If it does, the output is copied without any sort of identifying remarks to the control terminal of the message processor.

c. The command is executed asynchronously, using the background mechanism in the Bourne shell. Consequently, the processor does not wait for the command to finish before proceeding to its next task. If another message arrives before the command is complete, it simply invokes a separate copy of the command. In theory, the CMP could run out of processes on either the local or remote machine. If this happens, it will silently discard data until process table entries become available.

d. The command is executed with the effective user-id and group-id of the message processor. Normally, this is the system administrator account.

3.8.2.2 The Data Transfer Format.

Some client applications need to have certain information attached to the message data extracted by the processor: message originator, DTG, classification, etc. This information is called *envelope data*, because it pertains to the addressing of the message and not its contents. When the processor sends the output of a query to a client application, it encloses the output in an *output wrapper*, which contains the envelope data from the source message. The output and output wrapper together form a data package.

The processor obtains the envelope data for the output wrapper in one of two ways. The envelope data may be supplied directly to the processor through the input protocol (see Section 3.8.3).

It also processes the communication header lines on the input message to obtain envelope data. For example, it obtains envelope data from format lines 3, 5, 6, and 7 of JANAP-128 headers. If there is a conflict between the two sources (that is, the input protocol says one thing and the message header says another), then the processor will use the value from the input protocol. If neither source supplies a value for a particular envelope field, then it will use a default value.

3.8.2.2.1 Format Rules.

The information sent to client applications will always conform to the following format rules:

a. Wrapper lines always have a period as the first character on the line. Data lines never do.

b. All of the header lines will precede all of the data lines, which will precede all of the trailer lines.

c. The .END trailer line will always appear and will always be the last line in the package.

d. With the exception of the .END line, each wrapper line contains the value of one field in the envelope data. Wrapper lines begin with the initial period, the field name (MSGID, FROM, etc.), and a single space character (which is not part of the field value). The rest of the line is the value of the envelope item. This value will never contain more than 255 characters.

e. Header lines may appear in any order. Certain lines are optional; if they do not appear, then the value of the corresponding field is unknown.

3.8.2.2.2 Envelope Fields**.**

There are six envelope fields defined in the current output protocol. New envelope fields may be added to the protocol at any time, so client applications that process the wrapper lines must search for the envelope fields they require and ignore the fields they do not need. The defined envelope fields are:

a. The .MSGID line contains the message short name, extracted from field #1 of the MSGID set. (This is the only wrapper line containing information from the message text.)

b. The .FROM line contains the name of the message originator. For messages that have a JANAP 128 communications header, this field is extracted from the FM field, format line 6. The default value of this field is LOCAL.

c. The .TO line contains the name of the message addressee. For messages with a JANAP 128 header, this field is extracted from the TO field, format line 7. There is no default value for this field; if the value is unknown, the line is omitted from the wrapper.

d. The .DTG line contains the date and time at which the message was created. For messages with a JANAP 128 header, this field is extracted from the R field, format line 5. The field is 14 characters long, in the following format:

112233Z JAN 91

Characters 1 and 2 are day-of-month; characters 3 and 4 are hours (24-hour format); characters 5 and 6 are minutes; character 7 is a time-zone code; character 8 is a space; characters 9 through 11 are the month; character 12 is a space; characters 13 and 14 are the year.

e. The .CLASSIFICATION line indicates the security classification level of this message. For messages with a JANAP 128 header, this field is extracted from format line 3. The value of the field is a single character. The meanings of the possible values are defined in the JANAP 128 standard. There is no default value for this field; if the value is unknown, the line is omitted from the wrapper.

f. The .ERRORS line describes the result of the validation tests performed on this message. (See Section 3.6.1.6 for a description of message validation.) The value of this field is a number, possibly followed by text. The number is the only part of interest to automated systems; the subsequent text is a comment, for human readers only. The numeric part of the validation results must be interpreted as a collection of independent values represented by specific bit positions. These values are represented as follows:

   bit 0 (the least-significant bit) is set if the processor found errors anywhere in the message, and bit 1 is set if it found errors in the selected data elements.

All other bits are unused at present. However, client programs must not depend on these bits containing zero because they may be used in later parser release.

3.8.3 The Input Protocol

Incoming messages are submitted to the processor through the *input protocol*. This protocol includes the input mechanism. It also includes a special input format that allows the submitting program or user to supply envelope data without going to the trouble of constructing a valid message header.

3.8.3.1 The Message Input Mechanism.

The preferred way of submitting a message to the processor is to use the **mtf2parser** program. The arguments to this program are the names of files containing messages. The program can also read a single message from its standard input.

It is also possible to submit a message by copying it directly into a new file in the *mio* directory (see Section 3.4.2.2). The processor polls this directory for new files with a filename ending in the three characters *.in*. As soon as it finds such a file, it reads the contents as a new message and unlinks the file. To avoid a race between the processor and the copy operation, it is necessary to copy and then rename the new message file, for example:

```
cp message/usr/local/parser/mio/newmsg
mv/usr/local/parser/mio/new/usr/local/parser/mio/new.in
```

This method of submitting a message is discouraged because the process of polling the *mio* directory may be eliminated in a subsequent release.

3.8.3.2 The Message Input Format.

The message input format allows a message to be accompanied by envelope data without requiring this data to be placed into any particular communications protocol; for example, JANAP 128. Instead, the message text is optionally preceded by a simple, extensible header section that resembles the output wrapper in the output protocol. The input header appears as follows:

.FROM Plain Language Address (PLA)

.TO DHDIAZZ/SYJ

.DTG 011200Z APR 93

.CLASSIFICATION U

Each line in the header contains the value of one field in the message envelope data. Message header lines have the following specifications:

a. Header lines always have a period as the first character on the line. All of the header lines will precede all of the lines in the message text.

b. The identifier token (FROM, DTG, etc.) must be an exact match with one of the predefined message envelope fields. Case is significant.

c. There must be a space character following the identifier token. This character does not become part of the field's value.

d. The rest of the input line is the value of the field. In no case will the processor read more than the first 255 characters of the value. Excess characters are silently discarded.

e. The value of the DTG field must contain exactly 14 characters in the following format: DDHHMMZ SMMMSYY (where D = day; H = hour; M = minute; Z = time zone; S = space; M = month; S = space; Y = year)

f. Header lines may appear in any order. All header lines are optional; if they do not appear, then the value of the corresponding field is unknown. (A default value may be used, or a value may be obtained from the message's communication header.)

g. Header lines that do not satisfy these rules are silently discarded.

There are four header lines defined in the current input protocol. Of these, the **.FROM** and **.DTG** lines are used internally by the processor. These fields, together with the message type, form a unique identifier for the message. For example, messages are listed in the message journal by this identifier. Users must be careful to avoid submitting two messages with the same message

type, originator, and date-time group. The **.TO** and **.CLASSIFICATION** header lines are simply passed through to the output. Their values are not used internally.

The processor input header may be combined with messages that have JANAP 128 headers. In case of conflict between the processor header section and the message JANAP header lines, the values in the processor header will be used.

(Note: future releases of the CMP will contain a generic header. The generic header will contain data fields sufficient to allow the communications module to use which ever format is required and will present a consistent look and feel to the user.)

3.9 ERROR MESSAGES

The following are error messages that are incorporated into the output of a validated message. Note that error messages immediately follow the line error discovered. A total validated message will have no error message returned.

A problem with range validation.

ACTION: The range of the field contents was not validated because other types of errors were previously detected. Select the correct option and refer to the Message Generation (Section 4) Help text function for proper format and allowable range.

Bad classification redundancy in line 4.

ACTION: A message has been received in error. Notify originator to resend the message.

Bad date-time group in line 5.

ACTION: A message has been received in error. Notify originator to resend the message.

Bad day in line 5.

ACTION: A message has been received in error. Notify originator to resend the message.

Bad end of line 2.

ACTION: A message has been received in error. Notify originator to resend the message.

Bad hours in line 5.

ACTION: A message has been received in error. Notify originator to resend the message.

Bad minutes in line 5.

ACTION: A message has been received in error. Notify originator to resend the message.

Bad operating signal in line 4.

ACTION: A message has been received in error. Notify originator to resend the message.

Bad separator in line 4.

ACTION: A message has been received in error. Notify originator to resend the message.

Bad separator in line 5.

ACTION: A message has been received in error. Notify originator to resend the message.

Bad time zone in line 5.

ACTION: A message has been received in error. Notify originator to resend the message.

Bad year in line 5.

ACTION: A message has been received in error. Notify originator to resend the message.

Cannot determine FFIRN/FUDN.

ACTION: The MTF cannot determine the FFIRN/FUDN needed for validation. Check MIL-STD-6040 for correction contents.

Cannot divide up a composite field.

ACTION: The composite contents of the field cannot be divided into elements for validation. Select the correct option and refer to the Message Generation (Section 4) Help text function for correct format.

Cannot find S logic separator.

ACTION: Cannot find the separator between elements in the contents of this field. Select the correct option and refer to the Message Generation (Section 4) Help text function for correct format and contents.

Cannot find validation rules in data tables.

ACTION: The validation rules of the FFIRN/FUDN are not in the data tables. Select the correct option and refer to the Message Generation (Section 4) Help text function for correct format and contents.

Cannot identify first field in the composite.

ACTION: The first element of this composite field is not numeric. Select the correct option and refer to the Message Generation (Section 4) Help text function for correct format.

Cannot resection an un-sectioned message.

ACTION: A message has been received in error. Notify originator to resend the message.

Character not in legal set.

ACTION: The field contains an invalid character. Select the correct option and refer to the Message Generation (Section 4) Help text function for valid characters.

Checksum does not match.

ACTION: The checksum comparison of the running checksum total and the checksum element of the field has failed. Select the correct option and re-compute all numbers in the field.

Command line error.

ACTION: The data entered when initializing the system is incorrect. Usage is: parser [warmstart(comm)|coldstart(comm)][ui]

Composite field is too short.

ACTION: A composite element entered in the field is shorter than required. Select the correct option and refer to the Message Generation (Section 4) Help text function for correct format.

Decimal point prohibited in this numeric range.

ACTION: The field value does not contain a required decimal point. Select the correct option and refer to the Message Generation (Section 4) Help text function for the correct format.

Decimal point required in this numeric range.

ACTION: The field value does not contain a required decimal point. Select the correct option and refer to the Message Generation (Section 4) Help text function for the correct format.

Duplicate section.

ACTION: A message has been received in error. Notify originator to resend the message.

End of line 4 not found.

ACTION: A message has been received in error. Notify originator to resend the message.

End of line 5 not found.

ACTION: A message has been received in error. Notify originator to resend the message.

End of message not found.

ACTION: A message has been received in error. Notify originator to resend the message.

Field above alpha or numeric range.

ACTION: The value contained in the field is above the valid range. Select the correct option and refer to the Message Generation (Section 4) Help text function for the allowable range.

Field below alpha or numeric range.

ACTION: The value contained in the field is below the valid range. Select the correct option and refer to the Message Generation (Section 4) Help text function for the allowable range.

Field entry too long.

ACTION: Entry in the field is longer than the maximum allowed. Select the correct option and refer to the Message Generation (Section 4) Help text function for allowable length.

Field entry too short.

ACTION: Entry in the field is longer than the maximum allowed. Select the correct option and refer to the Message Generation (Section 4) Help text function for allowable length.

Field is empty.

ACTION: The contents of the field are missing. Select the correct option and refer to the Message Generation (Section 4) Help text function for correct contents.

Field prefix is bad.

ACTION: The prefix does not match the defined prefix according to the rules specified by the FFIRN/FUDN. Select the correct option and refer to the Message Generation (Section 4) Help text function for allowable values.

Invalid query specified.

ACTION: The syntax of the query text is incorrect. Reference the User's Manual for the correct syntax.

Line 11 not found.

ACTION: A message has been received in error. Notify originator to resend the message.

Line 13 not found.

ACTION: A message has been received in error. Notify originator to resend the message.

Line 2 not found.

ACTION: A message has been received in error. Notify originator to resend the message.

Message is sectioned and without line 5.

ACTION: A message has been received in error. Notify originator to resend the message.

Message is sectioned and without line 6.

ACTION: A message has been received in error. Notify originator to resend the message.

MSGID not found or message not in standard.

ACTION: A message has been received in error. Notify originator to resend the message.

Multiple sections in message section.

ACTION: A message has been received in error. Notify originator to resend the message.

No match on code list.

ACTION: The FFIRN/FUDN contains '-' instead of 'UNK'. MIL-STD-6040 requires 'UNK' to be used to denote missing data. If this is not the reason for the error, check MIL-STD-6040 for allowable codes.

Not in alphabetic sequence.

ACTION: Field entry is out of sequence. Select the correct option and refer to the Message Generation (Section 4) Help text function for the correct sequence.

Not left justified in field.

ACTION: The field is not left justified. Select the correct option and refer to the Message Generation (Section 4) Help text function for correct format.

Not right justified in field.

ACTION: The field entry is not right justified. Select the correct option and refer to the Message Generation (Section 4) Help text function for correct format.

Output file expected after -o flag.

ACTION: The next argument following -o must contain the file name of the output file.

Query file expected after -qf flag.

ACTION: A file name that contains the query text must follow -qf. Check the spelling of the file name.

Query string expected after -q flag.

ACTION: The next argument following -q must contain the text of the query. It is necessary to enclose the query in quotes.

Repeated codes not permitted.

ACTION: Repeated codes are not permitted. Select the correct option and refer to the Message Generation (Section 4) Help text function for correct format and contents.

Section count mismatch.

ACTION: A message has been received in error. Notify originator to resend the message.

Too few fractionals to the right of the decimal point.

ACTION: The field contains more digits to the right of the decimal point than are allowed. Select the correct option and refer to the Message Generation (Section 4) Help text function for the maximum allowable number of digits.

Too few fractionals to the right of the decimal point.

ACTION: The field does not contain the minimum number of digits to the right of the decimal point. Select the correct option and refer to the Message Generation (Section 4) Help text function for the minimum allowable number of digits.

Too few integers to the left of the decimal point.

ACTION: The field does not contain the minimum number of digits to the left of the decimal point. Select the correct option and refer to the Message Generation (Section 4) Help text function for the minimum allowable number of digits.

Too many integers to the left of the decimal point.

ACTION: The field contains more digits to the left of the decimal point. Select the correct option and refer to the Message Generation (Section 4) Help text function for the maximum allowable number of digits.

Unable to distinguish bound of first part.

ACTION: This field contains more than one variable length partition in the data item. Field contents cannot be analyzed. Select the correction option and refer to the JAMPS Help text function for correct format.

Usage: GenDraft[-v][inputfile]

ACTION: The syntax of the command has been incorrectly entered. Reference the User's Manual for the correct syntax.

Usage: mtfval[-vo][outputfile][inputfile]

ACTION: The syntax of the command has been incorrectly entered. Reference the User's Manual for the correct syntax.

USMTF MSGID not found.

ACTION: A message has been received in error. Notify originator to resend the message.

Validation logic code is bogus.

ACTION: Unknown elemental/composite or logic data exists in the field and cannot be validated. Select the correct option and refer to the Message Generation (Section 4) Help text function for correct format.

3.10 QUERY LANGUAGE SYNTAX

The query grammar can be summarized by the BNF notation listed below, in which the following notational conventions are observed:

a. [...] denotes 0 or 1 occurrence of contents
b. {...} denotes 1 or more occurrences of contents
c. | denotes alternatives
d. terminals are denoted using bold typeface as shown
e. <non_terminals> are denoted using the less-than and greater-than symbols as shown.

<query>::=[{<optional_clause>}]

<select_clause>[{<optional_clause>}]from_clause>

[{<optional_clause>}];|[{<optional_clause>}]<from_clause>[

{<optional_clause>}]<select_clause>[{<optional_clause>}];

<binary_op>::==|!=|>|<|>=|<=

<blank> ::=a single blank character

<bool_op>::=AND | OR

<cond_expr>::=<cond_expr>[<bool_op><cond_expr>]|

(<cond_expr>[<bool_op><cond_expr>])|<condition>

<condition>::=<message_element><binary_op operand>|

(<message_element><binary_op><operand>)|<message_element>

<unary_op>|(<message_element><unary_op>)

<digit>::=<sig_digit>|0
(display_clause> ::=FORMAT <target_format>
<err_clause>: :=VALIDATE<validate_level>
<FF> ::=<sig_integer>/<sig_integer>
<field_id> ::=.<sig_integer>
<from_clause>::=FROM<msg_expr>
<integer> ::={ <digit>}
<legal_char> ::=<letter>|<digit>|<USMTF_special_char>|<pattern_char>|blank>
<legal_msgid>: :=<USMTF_msgid>|*
<legal_setname> ::=<USMTF_setid>|*
<letter> ::=A|B|...|Z
<literal> ::= " {<legal_char>} "
<message_element>: ::=[{<seg_id>}][<set_id>[<field_id>]]
<message_element_list> ::=<message_element>[{,<message_element>}]
<msg> ::=DTG<binary_op><literal>|MSGID=<literal>|
MSGID !=<literal>|ORIG=<literal>|ORIG !=<literal>
<msg_expr> ::=<msg_expr>[<bool_op><msg_expr>](<msg_expr>[<bool_op><msg_expr>])|<msg>

<number> ::=<integer>|<real>
<operand> ::=<message_element>|<literal>|<number>|<FF>
<optional_clause> ::=<within_clause>|<where_clause>|<display_clause>|<err_clause>
<pattern_char> ::=*|%
<real> ::=[<integer>].<integer>|<integer>.[<integer>]
<seg_id> ::=[<legal_setname>]|[<sig_integer]|[]
<set_id> ::=<legal_setname>|(<sig_integer>)
<select_clause> ::=SELECT<message_element_list>
<sig_digit> ::= 1|2|...|9
<sig_integer> ::=<sig_digit>[{<digit>}]
<sig_legal_char> ::=<letter>|<digit>|<USMTF_special_char>
<target_format> ::=TEXT|JAMPS|ERRORS|TABLE|REPORT
<unary_op> ::=[!]EXIST
<USMTF_msgid> ::=<sig_legal_char>[{<legal_char>}]
<USMTF_setid> ::=<sig_legal_char>[{<legal_char>}]
<USMTF_special_char> ::=.|,|:|-|/|(|)|?
<validate_level> ::=NONE|EXTRACTED|MESSAGE|ALL
<where_clause> ::=WHERE<cond_expr>
<within_clause> ::=WITHIN<seg_id>[{<seg_id>}]

3.11 SHELL SYNTAX

The shell description can be summarized by the BNF notation listed below, in which the following notational conventions are observed:

a. [...] denotes 0 or 1 occurrence of contents
b. {...} denotes 1 or more occurrences of contents
c. | denotes alternatives
d. terminals are denoted using bold typeface as shown
e. <non_terminals> are denoted using the less-than and greater-than symbols as shown.

<shell> ::=[{<comment>}] [<defines>]{[{<comment>}]{<scope>}

[{<comment>}]}

<JQL_statement> ::=As formalized in Section 5.3
<begin> ::=<default_begin>|<defined_begin>
<begin_define> ::=#define start <defined_begin>;
<comment> ::=//[{<printchar>}]|/*[{<printchar>}]*/
<default_begin> ::={
<default_end> ::=}
<defined_begin> ::=<printchar>,except/and*
<defined_end> ::=<printchar>,except/and*
<defined_output> ::=PRINTER|SCREEN|"<text>"
<defines> ::=<begin_define>|<end_define>|<begin_define><end_define>|
<end_define><begin_define>
<end> ::=<default_end>|<defined_end>
<end_define> ::=#define end <defined_end>;
<output_define> ::=#define output<defined_output>;
<printchar> ::=any printable character
<scope> ::=[<defined_output>]<begin>[<text>][JQL:<JQL_statement>]
[<text>]<end>
<text> ::={<printchar>}

4 OUTBOUND MESSAGE PROCESSING

4.1 OVERVIEW

This section of the user's manual is designed to help you use the Message Generation Module. This paragraph is designed for users with varying degrees of message preparation and computer experience. Users of all experience levels should read through the rest of this section. After this overview, a plan for using the rest of the manual is presented, based on your level of experience.

This overview consists of the following subdivisions:

a. "**How to Use This Section**" describes what is contained in each subdivision of this section of the user's manual, along with a number of terms and conventions with which you should be familiar to use this manual most effectively.

b. "**What is the Message Generation Module**" explains formatted text messages and how they are created using this module.

c. "**What is New About the Message Generation Module**" gives an overview of the features, and introduces the graphical user interface.

d. "**What to do Next**" provides a game plan to follow for using the manual according to your experience level.

e. "**References and Standards**" is a list of sources for more information about message preparation.

4.2 HOW TO USE THIS SECTION

Detailed information on all functions and options are provided in Section 4.11 Tutorial - Preparing and Editing a Message. This paragraph explains the typographical conventions used in this section and explains what can be found in the rest of the manual.

4.2.1 Typographical Conventions

This section uses the following typographical conventions in the text to highlight important points and to distinguish user inputs from system output:

a. Button labels, menu names, menu options, dialog names, and data entry field labels are shown in Helvetica

b. Specific keys on the computer keyboard are identified by their key label in uppercase letters set in HELVETICA typeface. For example, the return key is denoted as RETURN.

c. Characters that must be entered by the user are shown in bold **Courier** typeface.

d. indicates that the steps that follow use the keyboard as the input device.

e. indicates that a toolbar button can be used to carry out an action.

f. **For Your Information**: A note like this supplies additional information or serves to clarify a point.

4.2.2 Active Functions

The outbound message processing software is in an evolving product, with some functionality not yet incorporated into this version of the software. Functions that are not practicable are noted by 1) function not occurring, 2) Pull-down menu options not operational ("grayed out"), or 3) a dialog box informing user that function is not implemented.

4.2.3 Contents of this Section

The rest of this section is divided into seven major paragraphs described briefly below.

a. "Installing and Configuring the Message Generation Module" (See Appendix A).

b. Paragraph 4.7: "Graphical User Interface" provides hands-on instructions for using the program interface.

c. Paragraph 4.8: "Main Displays and Help System" describes the two main displays used in the program and gives step-by-step instructions for using the help system.

d. Paragraph 4.9: "Tailoring Addresses, Originators, Drafters, and Releasers" explains how to use addresses effectively.

e. Paragraph 4.10: "Tutorial - Preparing a Message" provides step-by-step instructions which explain how to perform the basic functions of the Message Generation module.

f. Paragraph 4.11: "Advanced Features" explains the more advanced features, including customizing the program, message restoration, and importing/exporting messages.

4.3 WHAT IS THE MESSAGE GENERATION MODULE?

The Message Generation Module is a computer-based program that provides tools to aid in the preparation of formatted text messages such as MTF messages. It provides for creating and editing formatted text messages. The system can accommodate all messages in the USMTF

standard as well as other agreed-upon message standards which conform to the MTF format rules. Message standards may be customized to meet system requirements. The system also supports plain language address databases for message addresses but prefers to access and retrieve such data from a system level table.

4.3.1 Message Generation Module System Limits

TBD	The total number of saved messages the program will handle
TBD	The number of lines per free text set allowed
TBD	The total number of plain language addresses allowed in the ADR file
TBD	The number of TO and/or INFO sets allowed in the header message (for transmission only)
TBD	The total number of routing indicators allowed in any one AIG.

4.4 WHAT IS NEW ABOUT THE MESSAGE GENERATION MODULE

This module offers a "user friendly" user interface (GUI) which is capable of processing either Character Oriented Messages (COM) or translated Bit Oriented Messages (BOM). Furthermore, the CMP is capable of distinguishing the type of message, translating the data as required, and performing message generation or parsing of both types of messages from a single message data definition table. The user is no longer required to remember commands to perform tasks because all the commands needed can be found in menus and dialog boxes. Through simple point-and-clicks procedures using the mouse, a formatted text message or message which will be translated into a Bit Oriented Message can be created in a manner similar to using a word processor. In addition, on-line reference materials provide rules for creating formatted text messages, and a help system located at the bottom of each message template is available to assist users.

4.5 WHAT TO DO NEXT

The following paragraphs offer suggested steps for users of varying levels of experience.

a. Users Experienced with Message Editors

Even though it is not required to relearn the task of message preparation, you may need to learn how to use the Message Generation application. By following the steps below, you will be able to transfer smoothly your existing knowledge of message preparation.

(1) If the Message Generation Module is not already installed, follow the instructions, in Appendix A, to install and configure the module.

(2) Read through Paragraphs 4.7 and 4.8, and perform the examples provided using a computer. This will give hands-on experience using the new CMP GUI.

(3) Try to perform the tasks in the tutorial in paragraphs 4.9 and 4.10. If the steps of the tutorial are too detailed, try performing the tasks without the manual, and refer back to it when problems are encountered.

(4) Skip paragraphs 4.11 and 4.12 for now. These paragraphs will be useful later if it is necessary to perform more advanced tasks or if questions arise about a specific feature.

b. Users Experienced with Graphical User Interfaces

Since GUIs are already familiar, the user should not have much difficulty learning the module interface, which uses all the common GUI elements. Knowledge of message preparation, however, is necessary. It is suggested that the following steps be followed.

(1) If the Message Generation Module is not already installed, follow the instructions, in Appendix A, to install and configure the module.

(2) Skim through paragraph 4.7, paying close attention to the parts on mnemonics and keyboard accelerators. In paragraph 4.8, read about the two main windows used in the module and the On Line Help System. Open the MTF messages window in the Help menu to read about formatted text messages.

(3) Skip paragraphs 4.11 and 4.12 for now. These sections will be useful later if it is necessary to perform more advanced tasks or if there are questions about a specific feature.

c. Users Experienced with Message Editors and GUIs

This manual is a good reference for the user when trouble occurs. The following steps will ease the transition from a previous message preparation application to the Message Generation Module.

(1) If the Message Generation Module is not already installed, follow the instructions in Appendix A to install and configure the module.

(2) Skim through paragraph 4.7, paying close attention to the parts on mnemonics and keyboard accelerators. In paragraph 4.8, read about the two main windows used in the module and the On Line Help System. Open the MTF messages window in the Help menu to read about formatted text messages.

(3) Try to perform the tasks in the tutorial in paragraphs 4.9 and 4.10. If following the steps of the tutorial is too detailed, try performing the tasks without the manual, and refer back to it when problems are encountered.

(4) Skip paragraphs 4.11 and 4.12 for now. These sections will be useful later if there is a need to perform more advanced tasks or if questions arise about a specific feature.

d. Users New to Message Editors and GUIs

This manual assumes no prior knowledge of the Message Generation application. The following steps will lead you through the process of message preparation, while explaining the necessary computer conventions.

(1) If the Message Generation module is not already installed, follow the instructions in Appendix A to install and configure the module.

(2) Read through paragraph 4.7, and perform the examples provided using a computer. These examples will give hands-on experience using the new interface

(3) Follow the Tutorial in paragraphs 4.9 and 4.10 to learn step-by-step how to create a message using the Message Generation module.

(4) Skip paragraphs 4.11 and 4.12 for now. These sections will become useful later if it is necessary to perform more advanced tasks or if there are questions about a specific feature.

4.6 REFERENCES AND STANDARDS

The following references should aid you in the preparation of messages:

MIL-STD-6040, U.S. Message Text Formatting (USMTF) Program

Joint User's Handbook for Message Text Formats (JUH-MTF), AFP 102-2, DA PAM 25-7, JTC3AH 9000, OPNAV-P-942-1-86, NAVMC 2800, Revision 4.0, dated 1 October 1991.

4.7 GRAPHICAL USER INTERFACE

4.7.1 Overview

This section will explain the GUI used by the Message Generation module. The interface follows the Motif® GUI standard. This section contains the following subdivisions:

"Getting Started" covers the basics of using the mouse, windows, and buttons. Each topic is followed by an example to give you hands-on experience using the module.

"Selecting a Message" explains how to select a message from the Outgoing Message Window.

"Choosing Commands from a Menu" explains the steps involved in using the mouse to open menus and select commands from them.

"Working with Windows" covers all that you need to know about windows to use the program.

"Using Scroll Bars" explains what a scroll bar is and how to use it.

"Using Dialog Boxes" explains what a dialog box is and the elements that are commonly found in the dialog boxes.

4.7.2 Getting Started

To use the Message Generation Module you must ensure that the system is set up properly, see Appendix A. Use of a mouse or other pointing device is supported.

a. Launching the application

The Message Generation Module will be launched in accordance with the system into which it is integrated. As a stand-alone module, launching is accomplished in accordance with Appendix A.

b. Point and select

This module has a GUI, which means that you manipulate objects on the screen by using the mouse to "Point and Select" items. For example, to "open a message," first "point and click" the message to select it, then "point and select" the OKAY button to open the selected message. All operations performed use this "point and click" method.

c. Using the mouse

The Message Generation Module uses a graphical selection device called a mouse for some user input. When you move the mouse, a cursor on the screen moves in the same way.

(1) For best control, hold the mouse in your hand with your fingers close to the mouse buttons and with the cable pointing directly away from you.

(2) Watch the screen while you move the mouse on the mouse pad next to the keyboard. Every move that you make with the mouse moves the pointer or arrow on screen in the same direction.

(3) Watch the screen as you lift the mouse from the mouse pad. Notice that the pointer on the screen does not move. If you run out of room on the pad when moving the mouse, simply lift the mouse and place it back down on the pad in any spot where there is more room.

d. Using mouse buttons

The following chart lists conventions defined for the use of the mouse:

Left Button	Used for selecting, activating, and setting the location of the cursor.
Right Button	Used for displaying pop-up menus.
Press	Pressing the left mouse button and holding it down while the mouse remains stationary.
Release	Releasing the left mouse button after it has been held down for an operation.
Click	Pressing and then releasing the left mouse button in quick succession.
Double Click	Clicking the left mouse button twice in quick succession.
Drag	Moving the mouse while pressing the left mouse button.

e. Pointer shapes

In this module, the mouse pointer takes on different shapes, depending on where on the screen the pointer is located and what actions are possible. The table that follows describes the various cursors used and when they occur. Windows will be explained later. For now, think of windows as the work area available for a particular task.

4.7.3 Choosing Commands from Menus

The various conventions used in menus are described below. Menu selections are the principal means by which you communicate with the system.

"Cascading Menu Indicator" is a triangle pointing to the right. When a menu item with a cascading menu indicator is highlighted, an additional menu is displayed to the right of the selected menu item (a cascading menu).

"Dialog Box Indicator" is three dots (e.g., Delete...) following a menu item. A dialog box will be displayed upon selection of the item. The dialog box will allow you to make choices about how you would like the command to be carried out. Items with no dots produce an immediate action when selected but in some cases, the program asks for confirmation before continuing.

"Mnemonics" are used to navigate through menus using only the keyboard. Each mnemonic character is underlined in the menu title or menu item. In this program, all menus as well as the titles of pull-down menus on the menu bar have mnemonics associated with them.

"Accelerator Key" is a key or key combination that invokes a command regardless of cursor location. Accelerators are most commonly used to activate menu items. The accelerator keys are displayed to the right of menu items. Sometimes the accelerator key is a combination of a letter and the Meta key on yours keyboard. On some keyboards the Meta key is labeled with a diamond.

Along the top of a main window, in the menu bar, are the titles of several menus. A module menu can be seen in the following figure.

a. To choose a menu command

(1) Move the mouse pointer to the menu title File and click the left mouse button.

Pressing the mouse button with the pointer on a menu title highlights the title, and a pull-down menu appears. This is called "pulling down" a menu. When the mouse button is released, the menu remains pulled down. Menus contain items that execute commands or open dialog boxes.

(2) Select the menu items from the menu bar by clicking on the menu item of your choice.

To close a pull-down menu without making any choice, move the pointer off the menu and click the left mouse button. The pull-down menu will close and the highlighted menu title will return to normal.

For Your Information: There is a second method of selecting an item from a menu. It is similar to the method described above except that you keep the mouse button pressed between steps 1 and 2. To select a Menu item using this method, press the left mouse button on the menu title, drag the cursor to the menu item of your choice, then release the mouse button.

b. Mnemonics

All menus have mnemonics to support users who do not have a mouse or would prefer not to use a mouse. A mnemonic is a single character that provides a quick way to make a selection from the keyboard. In this program, all menu items, as well as the titles of pull-down menus on the menu bar have mnemonics associated with them. Each mnemonic is a single underlined character in the menu title or menu item. It is often the initial letter of the selection. When an initial letter cannot be used, as in the case where two selections begin with the same letter, another letter in the selection name is chosen.

To open a menu using the keyboard mnemonic, press the ALT key (on SUN platform this is a meta key) and the underlined mnemonic letter for that menu simultaneously. The menu will be opened, and any selection in that menu can be made by directly typing the appropriate mnemonic letter for that selection. Note that the ALT key need not be pressed to use a mnemonic once the menu is open.

To select a menu item (without using the mouse)

(1) While holding down the ALT key, press **F**. This pulls down the menu, showing you the contents.

(2) Press **S**.

This executes the Save command.

For Your Information: This two-step process of opening a menu and selecting a menu item is convenient if you need to search for a menu item. However, if you already know which command you want, using accelerator keys is quicker.

c. Accelerator keys

Accelerator keys are shortcut keys that allow you to execute a command by pressing a key combination, usually the CONTROL key and the first letter of the command. A difference between mnemonics and accelerators is that only accelerators can be activated without first opening a menu.

Action	Mnemonic	Accelerator
Address List (PLA)	A	----
Append Field	----	----
Application Preferences	A	----
Clear	e	<Ctrl-R>
Close	C	<Ctrl-W>
Context-Sensitive Help	C	<Shift-Help>
Copy	C	<Ctrl-C>
Customize	u	----
Cut	t	<Ctrl-X>
Delete	D	----
Deselect All	----	<Ctrl-\>
Down One Level	D	<Meta-J>
Drafter List	D	----
Exit	X	<Ctrl-Q>
Export To	E	----
Field	F	----
Field Group	G	----
Group List	G	----
Header	H	<Meta-H>
Header List	L	----
Help Overview	H	F1
Import From	I	----
Insert Field	F	----
Insert Set	S	----
Interface Help	J	----
Maximize	X	<Meta-F10>
Message Disk	D	<Ctrl-D>
Minimize	N	<Alt-F9>
Move	M	<Alt-F7>
New Address	N	<Ctrl-A>
New DD173 Header	----	----
New JANAP Header	H	<Ctrl-N>
New [Message Standard]	N	----
Next Field Alternate	A	<Meta-A>
Open	O	<Ctrl-O>
Originators	O	----
Paste	P	<Ctrl-V>
Preview	v	<Meta-I>

Print	P	<Ctrl-P>
Printer Setup	P	----
Reformat	m	----
Releaser List	R	----
Remove Header	----	----
Repeat	R	cascade options
Restore Message	R	----
Save As	A	----
Save	S	<Ctrl-S>
Segment	m	----
Select All	----	<Ctrl-/>
Set	S	----
Show Field/Set/Message Errors	F	----
Show Field Codes	C	<Ctrl-F>
Show Field Maps	M	----
Show Segment Bars	B	----
Show Sets Expanded	S	----
Spelling	L	<Ctrl-L>
Structural Errors	S	----
Templates	T	----
Tutorial	T	----
Undo	U	<Ctrl-Z>
Up One Level	U	<Meta-U>
USMTF Messages Browser	U	----

To use an accelerator key to save, press S while holding down the CONTROL key. This executes the Save command.

d. Cascading menus

A cascading menu is a sub-menu that pops up when a menu item is highlighted.

4.7.4 Working with Windows

A window is an enclosed work area for a particular task. This module uses two main windows, which are explained in paragraph 4.9.1. Windows can be moved, resized, scrolled through, and closed. Multiple windows can be displayed on the screen at once, allowing you to refer to two windows while keeping both in view. To use a window, it must be active. When more than one window is open at once, the window that lies on top of all others is the active one. To change the

active window, just click on any part of the window desired or select the desired window from the Windows menu.

a. To resize a window:

(1) From the Help menu, select the Interface Help menu item. This is just to give you a window to use for this example. The techniques presented here can be applied to all windows.

(2) Move the pointer to the lower right corner of the Help window. The cursor will change from the pointer to a window resize cursor.

The arrow cursor indicates that the window can be resized. The window can only be resized when the cursor is an arrow.

For Your Information: If the Help window suddenly disappears, it is probably because you clicked on the Message Edit Window, bringing it to the front. To get the help window back, perform the first step again.

(3) While the cursor is still an arrow, press the mouse button. While the button is pressed, move the mouse.

You are now re-sizing the window. As you move the mouse around, only the outline of the window will stretch.

(4) Release the mouse button.

When the mouse button is released, the window is resized to the shape of the outline you created.

b. To move a window:

Drag the title bar of the window to its new location

c. To make a window active:

Click on any part of the window. If no parts of the window are showing on the screen, choose the window from the Windows menu.

d. Window control menu

The Window Control menu is used to display a list of window actions. The Window Control menu button is located in the upper left corner of most windows. Pressing the window menu button activates the menu and presents all or some subset of the following options: Restore, Move, Size, Minimize, Maximize, Lower, and Close.

(1) To open the window control menu, press the upper left corner of the window.

Restore The Restore option restores the minimized or maximized window to its previous size and location.

Move The Move option moves a window around the work space.

Size The Size option changes the height and width of the window in the direction indicated by the pointer.

Minimize The Minimize option changes a window into an icon, which is a small pictorial representation of the window.

Maximize The Maximize option enlarges a window to its maximum size.

Lower The Lower option moves a window behind all other windows displayed on the screen.

Close The Close option closes a window and removes it from the workspace.

(2) To close a window:

Select Close from the Window Control menu.

For Your Information: Double clicking on the Window Control menu button closes the window.

4.7.5 Using Scroll Bars

Windows often cannot show all of the information that they contain on screen at once. The scroll bar is used to view the parts of the window that will not fit on the screen. Scroll bars are only active when there is more information than can fit in the given area.

a. To scroll through text:

Resize the help window to a smaller size, so that all of the text is no longer in the window.

b. To move line-by-line:

Click the upper or lower stepper arrows. Notice that more text comes into view. This is called scrolling.

c. To move page-by-page.

Click in the trough above or below the slider as pictured above. Notice that the text scrolls more rapidly.

d. To move quickly to the beginning or the end:

Click and drag the slider up or down.

For Your Information: The slider moves as you scroll through the message. When the slider is at the top of the trough, you are at the beginning of the message; when the slider is at the end of the trough, you are at the end of the message.

4.7.6 Paned Windows

A paned window is a window that contains areas that can be resized. The window is part of the Message Edit window. This paned window contains an upper and lower pane, separated by a horizontal split bar.

To resize a pane in a paned window:

Click on the horizontal split bar button and drag the bar up or down to resize the panes as needed. Notice that the smallest size the header pane (the upper pane) can achieve is one line.

4.7.7 Using Dialog Boxes and Controls

A dialog box is a specialized type of window that simply gives you alternative choices. For example, when the menu item Print... is chosen, there are some preferences that you can select.

a. To see the Print Message dialog box, select the Print menu item

From the File menu, in the Message Edit Window.

The following buttons are common to many dialog boxes:

(1) **OK Button** The OK button applies the settings that were changed and closes the dialog box. Sometimes, as in the Print dialog box above, an action button is substituted (e.g., the word Print).

(2) **Cancel Button** The Cancel button resets settings to their states before the dialog box was opened, then closes the dialog box.

(3) **Help Button** The help button opens a help topic for the current dialog box.

For Your Information: Some dialog boxes contain default buttons. They are surrounded by a shadow. The Print button, shown above, is an example of a default button. These buttons can be activated by pressing the RETURN key, as well as by clicking the mouse.

b. Check boxes

Check boxes are used to toggle items "on" or "off." Multiple options in the check button group are allowed to be "on" simultaneously.

To select items in a check box:

Click on the squares to the left of all of the items desired to be toggled on.

For Your Information: You may select as many check box items as you need. Clicking on a check box toggles it on and off. To turn a check box off, click on it.

c. Radio or option buttons

Radio buttons (also called option buttons) are also used to toggle items 'on' and 'off.' Unlike check boxes, however, only one item may be selected from a grouping of radio buttons. Radio buttons work like buttons on a car radio. You can only choose one radio station at a time.

To select an item from a group of radio buttons:

(1) Click on the diamond (or circle) to the left of the item desired. This will toggle the selected item on, and will toggle the rest of the radio buttons in that group off.

(2) To close the Print Message dialog box without printing, click on the Cancel button.

d. Data entry fields

A data entry field is an area in a window where text can be entered by the user.

To enter data into a data entry field:

(1) From the Address Menu, select the menu item New Address...

This will open the New Address dialog box to use as an example for text entry.

(2) Position the cursor over the darker area to the right of the Long Address: field label.

This moves the cursor to where you want to insert text.

(3) While the cursor is still an I-beam, click the mouse button.

A flashing vertical bar can now be seen in the Long Address: field. Any typing you do now will be inserted under the flashing bar.

(4) Now enter the address HQ 1912 CSGP Langley AFM VA

To delete any unwanted characters, use the BACKSPACE key.

For Your Information: Text fields that do not have a shadowed box around them are "display only." The system places text in these fields for you to read, but you cannot edit the text. When the cursor is moved over a "display only" text field, the cursor will not change to an I-beam.

e. To copy field data: ((This function is not implemented at this time))

(1) To select the text to be copied, drag the cursor over the text. The text is now highlighted, indicating that it is selected.

(2) From the Edit menu, select the Copy item. This copies the selected text to a buffer.

(3) Click on the text field to which you want to copy the text.

(4) From the Edit menu, select the Paste item. This copies the text from the buffer to the current field.

f. To move field data from one field to another: ((This function is not implemented at this time))

(1) Select the text to be moved by dragging the cursor over the text. The text is now highlighted, indicating that it is selected.

(2) From the Edit menu, select the Cut item. This removes the highlighted field from the current field and places it in a buffer.

(3) Click on the text field to which you want to move the text.

(4) From the Edit menu, select the Paste item. This copies the text from the buffer to the current field.

g. Drop down combo boxes

A drop down combo box is a combined text entry and menu field. A menu appears when the triangle graphic button to the right of a field is pressed. Menu items can be selected by dragging the cursor to the desired item. The menu item then replaces the text in the field. Sometimes you can also enter data directly into the text box after clicking in the field to make it active.

To use a drop down combo box:

(1) Click on the triangle graphic button with the left mouse button, which opens the drop down list.

With the left mouse button, drag the cursor to the item desired.

Select Item and move out of box. Item is now selected.

For Your Information: The menu can also be posted (i.e., opened) by clicking once on the triangle button.

h. Tool Bar - Message Edit Window has push buttons for:

Save the current message.

Print the current message.

Undo the last operation. ((This function is not implemented at this time))

Cut and store selection into buffer. ((This function is not implemented at this time))

Copy selection into buffer. ((This function is not implemented at this time))

Pastes contents of buffer into current location. ((This function is not implemented at this time))

Repeats selected item.

Displays message header.

Toggles segment bars on and off.

Toggles field maps in the field on and off. ((This function is not implemented at this time))

Shows message field and set errors and shows structural errors.

Opens spelling dialog box. ((This function is not implemented at this time))

4.8 MAIN DISPLAYS AND HELP SYSTEM

NOTE: The On Line Help function is not available at this time.

4.8.1 Overview of Main Displays and Help System

Message Generation Main Displays describes the two main windows that you will use with the program.

Using the On Line Help System explains how to use the help system provided with the Message Generation Module.

4.8.2 Main Displays

The Message Generation application has two main windows, the CMP User Interface (CUI) Window, and the Message Edit Window. The CUI is the window used for message management, while the Message Edit Window is used for editing the contents of individual messages.

a. CMP User Interface

(1) Menu Bar The Menu Bar contains a list of the following menus, each of which contains a list of commands:

The **File menu** contains a list of options that affect messages in their entirety.

File Options include the ability to generate new messages and new Auto-Fill messages. Messages may be saved and/or archived from the pull down window. Messages may be printed from this option.

The Message menu enables editing and reviewing of messages.

(2) Tool Bar The tool bar provides quick access to commands that are frequently used. The tool bar is positioned horizontally at the top of the program screen, just below the menu. When a push button from the Tool Bar is clicked on, the action represented by that push button occurs, such as;

Launching the message generation window.

Creating messages from data taken from a database from predefined conditions.

Saving the message.

Printing the message.

Deleting messages.

Previewing the message in text form.

Edits selected message using the message generation window.

Accesses message address header.

Sends messages to communications module for retransmission.

Forwards messages to addresses.

Enables user to reply to selected messages.

Filters messages based upon predetermined criteria.

Attaches memorandum to saved messages.

(3)	Incoming Messages List	This section provides a list of messages that have been received.
(4)	Temporary Messages List	This section provides a list of messages currently being reviewed or edited.
(5)	Outgoing Message List	This section provides a list of messages that have been released to the communications module.
(6)	Column Headings	"**NAME**" displays the message identification.

"**Source"** displays the message originator.

"Destination" displays the message destination. If message has more than one destination, the number of destinations will be displayed in () and the 1st destination will be displayed.

"DTG" displays the message date-time-group.

"**Class**" displays the message classification.

"Precedence" displays the message handling precedence.

"Status" displays message status (e.g., read or unread) the scroll bar becomes active and you can scroll the message list. Horizontal message bars allow you to see text that is wider than the column width.

b. Message Edit Window

(1) "**Menu Bar**" contains a list of menus, each of which contains a list of commands.

(2) "**File menu**" contains a list of options that affect messages in their entirety.

(3) "**Edit menu**" provides a list of options for cutting, pasting, copying, repeating, and inserting text, fields, sets, and segments. It also contains header editing functions.

(4) "**View menu**" allows you to show segment bars, expanded sets, and field maps in the fields. It also allows the user to toggle to the next alternate field and to show field codes.

(5) "**Tools menu**" allows you to show errors, spell-check, and restore messages. This menu also allows the user to find, replace, and set preferences.

(6) "**Windows menu**" displays the titles of windows that are currently open in the application. (NOTE: NOT AVAILABLE AT THIS TIME).

(7) "**Help menu**" provides access to the different types of on-line help information.

(NOTE: NOT AVAILABLE AT THIS TIME).

(8) "**Tool Bar**" gives quick access to commands that are used often. When a push button from the Tool Bar is clicked on, the action represented by the push button takes place. From left to right, the push buttons are Save, Print, Preview, , Repeat, Edit, Header, Segment Bar, and Error/SNVO_ID. For a complete explanation of each push button, see paragraph 4.7.9. (NOTE: THE UNDO, CUT, COPY, PASTE, FIELD MAP,AND SPELLING ARE NOT AVAILABLE AT THIS TIME).

(9) "**Message Header**" displays the header information, if any, for the current message.

(10) "**Message Body**" displays the segments, sets, fields, and field data of the current message.

c. Use of Color

Color is used to highlight parts of the message and to add meaning to status lines and set identifiers. Colors used in message templates are shown below.

Color	Usage
Red	Mandatory sets and fields, and conditional mandatory sets. The error indicator in the status error.
Green	Operationally determined sets and fields, conditional sets, unclassified message indicator.
Yellow	Conditional set identifiers
Field turns Gray	Background color of fields with errors
Green	In the help system, clicking on green underlined text will display a related help topic.
Green	In the help system, clicking on text like this will show a pop-up window with the definition of the word(s) selected.

Note: Fields will appear black until tabbed into; then they change to red or green based on the occurrence category (mandatory or optional) of the field.

d. Set/Field/Command/Error Status Area

The status area provides information about the active field, set, and command (if cursor is in the menu area).

(1) "**Set ID**" displays the color-coded set identifier of the active set.

(2) "**Set Flags**" displays the color-coded single letter set description.

(3) "**Field Flag**" displays the color-coded single letter field description of current field.

(4) "**Error Flag**" displays a single letter indicating whether the current field contains an error.

(5) "**Set Name**" displays the full set name.

(6) "**Field Description**" displays the field name.

Character	**Occurrence Category**
O	Operationally Determined Field (Optional)

M	Mandatory Field
C	Conditional Field
R	Repeatable Field
Alt	Alternate Content Field
E	Error in Field
O	Operationally Determined Set (Optional)
M	Mandatory Set
C	Conditional Set
R	Repeatable Set

(7) "**Set Number**" displays the number of the set currently displayed at the top of the message window.

(8) "**Error Information**" displays the current field error.

(9) **"Field Range Information"** displays the field map and range information of current field.

(10) "**Element Selected**" displays the part of the current message that is selected.

One very important part of the user interface is the help system. Much of the information contained in this user's manual is also contained in the help system.

4.8.3 Using the Help System (NOTE: NOT AVAILABLE AT THIS TIME.)

The Message Generation Module has a complete on-line reference tool that can be accessed at any time. The help system is especially useful when information is needed quickly. The help system contains help on every menu item, dialog box, and window. It also contains step-by-step procedures for completing most basic tasks in the module. The help windows can be resized and moved so that you can view the help system while performing your message preparation tasks. The help system can also be used to get more detailed explanations of fields, sets, and messages. This information continually updates itself as you move through fields in your message.

a. Getting Help

(1) Most dialog boxes have a Help button. By pressing this button, the Help system will open to the topic that explains that particular dialog box. Once the help system is opened, the user can browse through it to find information on any chosen topic.

(2) The help system can also be accessed through the Help menu. Help Overview gives a summary of the various types of help offered by the help system.

b. Help Overview

The Help Overview menu item gives a summary of the on-line help system as well as help on using the help system.

(1) To show the Help Overview

From the Help menu (in either of the program's two main windows), choose Help Overview.

The help menus are slightly different depending on which main window you are currently using. The Message edit window has an additional help menu called Real Time Help, which has context-sensitive help on the message currently being viewed.

(A) "**Real Time Help**" opens a window that gives field-, set-, and message-dependent help on the active field. The window will remain on top of all other windows, even when other windows are activated. This type of window is referred to as a floating window.

(B) "**User Interface Help**" contains help concerning interface elements and conventions used by the Message Generator.

(C) "**Tutorial**" presents a list of step-by-step procedures for message preparation.

(D) "**USMTF Messages**" provides help on USMTF message format rules, including general instructions on message preparation. USMTF Message Help also contains the USMTF Message Browser, which contains the purpose of each of the message templates installed in the program (this feature should ultimately be expanded to include all messages supported).

c. User Interface Help

The interface help explains how to use the graphical elements that are part of the program interface. Interface Help also explains every window that is part of this module.

(1) To show the Interface Help:

From the Help menu, choose Interface Help.

The table of contents of the Interface Help is now shown. Notice that the text is green and underlined. Clicking on green, underlined text brings the user to the section being described. If you click on Getting Started, the help system will show information on Getting Started with using the program.

d. Moving around in Help

Help buttons are located across the top of every window except Help Overview. These help the user to navigate through the help system by;

display of contents of the help menu item being used,

allowing the user to search for a topic by a keyword,

showing the last topic you looked at,

displaying a list of topics already viewed,

displaying the previous topic in a series of topics,

displaying the next topic in a series of topics.

e. Keeping Help on Top of all other Windows is useful if you need to see the help as you perform tasks.

To keep help on top:

From the Help menu in the Interface Help window, select Always on Top. A toggle button to the left of the menu item indicates that the menu item has been selected.

f. Viewing a Message and Help Together

You may want to see the Help Window and the Message window together. To do this, you may need to resize and/or move the help window.

To resize the help window:

(1) To resize the help window, click and drag the lower right corner of the help window. For more detailed information about re-sizing a window, see paragraph 4.7.5, Working with Windows.

(2) To move the help window, click on the title bar of the help window and drag the window to its new location.

g. Scrolling Through a Help Topic

If the information contained in a Help Topic cannot fit on one screen, the scroll bar can be used to look through the information.

To scroll through the topic line-by-line:

Click on the scroll arrows in the scroll bar.

To scroll through the topic page-by-page:

Click above or below the scroll box in the scroll bar.

h. Searching for a Help Topic

You can search through the help system quickly by using a keyword search. This feature acts like an index in a paper manual.

To search for a help topic:

(1) From the Help buttons, select the Search button. This will bring up the search dialog box.

(2) Type a word or phrase to be searched for into the text box. You can also scroll through the upper list box of topics.

(3) Press the Show Topics button. This will fill the lower list box with topics that reference the selected topic.

(4) Select the desired sub-topic from the lower list.

(5) Press the Go To button. This will take you directly to the topic in the help system.

For Your Information: When you are using the program, many of the dialog boxes contain Help buttons. Pressing the buttons will automatically bring up help on the appropriate topic.

i. Defining and Using Bookmarks

Bookmarks allow the user to put place holders in the help system, making it easier to toggle between them. This is especially useful for topics referred to often.

To place a bookmark in the topic you are currently using:

(1) From the Bookmark menu, select Define.

(2) A dialog box will appear with a suggested Bookmark Name highlighted. To change the name to something else, type the name you would prefer. Press OK.

(3) The Bookmark will now appear at the bottom of the Bookmark menu.

To go to a bookmark:

(1) Select the Bookmark name from the Bookmark menu.

This will bring you to the topic associated with the bookmark. You can also type the number that corresponds to the bookmark in the Bookmark menu to jump quickly to the bookmark.

To remove a bookmark:

(1) From the Bookmark menu, select Define.

(2) From the list of Bookmarks, select the Bookmark to remove.

(3) Select the Delete button.

The bookmark is now removed.

j. Annotating a Help Topic

The annotate option allows you to add your own comments to a help topic.

To add your own comments to a topic:

(1) From the Edit menu, select Annotate.

A dialog box will pop-up, allowing you to type in text.

(2) Type in your comment.

(3) Select the Save button.

To see your annotation:

Press the green paper clip next to the title of the help topic.

k. Using Real Time Help

Real time help provides help on the particular field, set, or message that is currently being used. The Real time help changes when moving from field to field, from set to set, and from message to message. Real time help is shown in a separate window that floats on top of all other windows. When Real time Help is selected from the Help menu in the Message Edit Window, the Real Time Help is toggled "on." Like the status area, this window updates as you move from one field to another.

To turn on Real Time Help:

(1) Open a Message Edit Window.

(2) From the Help menu (on the Message Edit Window, not within the help system), select Real Time Help. This toggles the Real Time Help on and off. If you now open the help menu, you will notice that the toggle button to the left of the menu item Real Time Help is black, indicating that it is toggled on.

To turn on Field help:

(1) From the toolbar in the Real Time Help window, click on the Field button. This shows the Field Help in the real time window.

(2) Now click on one field in your message and then tab to another field. Notice that the Field help changes when you tab into another field, giving you help on the new field.

For Your Information: To get the other types of real time help, press the appropriate button located across the top of the real time help window. Try pressing the Message , Set, Example, and Notes buttons to see the kinds of information you get for each of these.

To turn off Real Time Help:

(1) From the Help menu (on the Message Edit Window, not within the help system), select Real Time Help. This toggles the Real Time Help off (See also paragraph 4.8.3).

I. Opening the Tutorial

The tutorial part of the help system explains the step-by-step procedures for message preparation.

To open the on-line tutorial:

From the Help menu in the Message Edit Window, select Tutorial. This opens the contents topic screen of the Tutorial.

m. Opening the MTF Messages Help

The MTF Messages portion of the help system provides help on MTF messages format rules, including general instructions on message preparation. MTF Message Help also contains the MTF Message Browser (currently focused on USMTF only), which contains the purposes of the message templates installed in the program.

To open the MTF Messages help window:

From the Help menu in the Message Edit Window, select MTF Messages. This opens the contents topic screen of the MTF Messages help.

n. Exiting the MTF Message Help

To close the MTF Message help window:

From the File menu in the Help window, select Exit.

4.9 TAILORING ADDRESSES

4.9.1 Overview

"Working with Addresses" explains how to display, add, edit, delete, and group addresses.

"Working with Drafters and Releasers" explains how to display, add, edit, and delete drafters and releasers. ((This feature is currently partially implemented.))

4.9.2 Working with Addresses

The Message Generation Module is pre-configured to allow you to produce messages immediately after installation has been completed. However, one can reduce the amount of time required to produce messages if you further configure the module to meet your specific needs. This section will show you how to display and add to the list of the addresses.

a. Listing Addresses

The address contained on your hard drive can be listed and edited.

To display the list of addresses:

(1) From the "Address Menu" choose "Address List (PLA)". The address menu is available in the Message Edit Window.

This displays the following dialog box, which shows a scrolling list of addresses and groups, and provides access to address management functions. Groups are listed first in the list, followed by addresses.

If there are many addresses, the vertical scroll bar can be used to see additional addresses. The horizontal scroll bars can be used to see information about an individual address that may not fit in the area provided.

"Long Address" (more commonly known as PLA) is the long address portion of the address.

"Office Symbols" is the office symbols portion of the address.

"Routing" is the routing portion of the address.

"New..." displays a dialog box that allows you to add an address to the list.

"Import..." opens the Import Address PLA dialog box, which allows you to import addresses.

"Edit..." displays a dialog box containing information on the currently selected address, allowing you to edit a selected address or group.

"New Group" opens the New Group dialog box, which allows you to select addresses to be included in a group. Groups allow you to send a message easily to a common group of recipients.

"Delete..." deletes any addresses or groups that are selected.

"Close" closes the address list window.

b. Adding Addresses

Addresses and office symbols can be added to your address list.

To add an address:

(1) From the Address List dialog box choose the New button. This displays the New Address dialog box.

Alternate Method: Another method of displaying the New Address dialog box is to choose New Address from the Address menu.

(2) Type **1234 ABCD Langley AFB VA,** into the Long Address text field.

(3) To get to the next text field, press the TAB key or click in it with your mouse.

The TAB key moves the cursor to the next text field. The tabbing order is from left to right and then down.

(4) Type the first office symbol **ABC**, into the Office Symbol text field.

(5) To add the Office Symbol to the address, click on the Add>> button.

To add additional office symbols, repeat the step above. If you make a mistake and add an incorrect office symbol to the address or want to remove one, highlight the incorrect office symbol in the scrolling box, then press the <<Remove button.

(6) You can specify one address to serve as the default originator by clicking in the Set as default originator check box while the address is selected. The address chosen will be used as the originator for messages unless the program is otherwise instructed.

The address is now complete. You now have the following options:

"Close" will close the dialog box. If you have not saved recent changes, you will be prompted to save.

"Save" will save the address.

"Save and Clear" will save the address and clear the fields so you can enter another address.

"Clear" will clear the address fields without saving changes.

(8) For now, if all of the information is correct, press the Save button and the Close button.

c. Editing Addresses

Addresses can be edited using the module to add or remove office symbols and also perform other editing tasks.

To edit an address:

(1) From the Address menu, choose Address List (PLA)...

(2) In the Address List dialog box, select the address to be edited and choose the Edit button. This displays the following dialog box is displayed:

(3) This dialog box works in the same way as the New Address dialog box. Use OK to save changes and Cancel to ignore changes.

d. Deleting Addresses

To delete an address:

(1) From the Address menu, choose Address List (PLA)...

(2) From the Address List dialog box, select the address to be deleted by clicking.

(3) To delete the selected address, press the Delete... button.

(4) The dialog box above will appear. Press OK to delete the address. If a group name had been selected, the group would have been deleted.

e. Grouping Addresses

Addresses that are used for a similar purpose can be grouped together using the Group List... menu item or the New Group button in the Address List dialog box.

To create a group of addresses:

(1) From the Address menu, choose Address List (PLA)...

(2) From the Address List (PLA) dialog box press the New Group... button.

(3) From the New Group dialog box select the addresses and corresponding office symbols from the Address and Office Symbol scrolling list to be added to the group.

Notice that when different addresses are selected, the office symbols change. The office symbols shown are only those associated with the highlighted address. At this point you can select certain office symbols associated with the highlighted address to be added to the group.

(4) To select Office Symbols to be added, select the Office Symbols from the Office Symbols scrolling list.

(5) To add the selections to a group, press the Add button.

(6) Repeat steps 3 through 5 as many times as needed to add the addresses to the group.

(7) To name the address group, click the Group Name: text box, and type in a meaningful name for the grouping.

(8) Press the OK button to accept the grouping.

To remove an address from a group:

(1) From the Addresses Selected for Group scrolling list, select the address to be removed.

(2) To remove the address, press the OK button.

Notice that the address is no longer in the Addresses Selected for Group scrolling list.

To remove just an office symbol:

(1) From the Address Selected for Group scrolling list, select the address that contains the Office Symbol to be removed.

(2) From the Office Symbol scrolling list, select the office symbol to be removed.

(3) To remove the office symbol, press the Remove OS button.

To edit group lists:

(1) From the Address menu, choose Group List

Alternate Method: From the Address List dialog box, select a group and press the Edit button.

The Group List dialog box above will appear. From this dialog box, you can also create new groups and delete outdated groups.

(2) To Edit a group list, select the group name from the Group Name scrolling list and then press the Edit button.

This opens a dialog box similar to the New Group dialog box, which works in the same way. (See "Grouping addresses" for more information.)

4.9.3 Working with Drafters and Releasers

Drafter and releaser information is needed when creating DD173 forms and headers. Here we will explain how to enter drafter and releaser information. Later, when the information is needed to populate headers, you can select the information from a list. The steps are essentially the same for working with drafters and releasers.

a. Listing Drafters

The drafters previously added and saved can be listed and edited.

To see a list of the drafters:

(1) From the Address menu, select Drafter List...

This opens the Drafter List box below.

b. Adding Drafters

To add a drafter:

(1) From the Address menu, select Drafter List...

This opens the Drafter List dialog box.

(2) From the Drafter List dialog box, select the New... button.

This opens the dialog box below.

The Name/Title field can accept up to 34 characters, and the Office Symbol/Phone field accepts up to 34 characters.

The Releaser Dialog box is accessed from the New button on the Releaser List dialog box. The Releaser dialog box is slightly different from the Drafter dialog box. It combines the fields Name/Titles and Office Symbol/Phone into one field. The maximum number of characters that can be entered is 32.

(3) Enter the appropriate information in each field. To continue entering information about other new drafters or releasers, press Save and then Clear. To save and close the dialog box, press OK.

c. Editing Drafter Information

To edit a drafter:

(1) From the Address menu, select Drafter List...

This opens the Edit Drafter dialog box.

(2) To choose a draft to be edited, select one from the Name list.

(3) From the Drafter List dialog box, select the Edit... button.

This opens the dialog box below.

(4) Make changes and press OK to save and close the dialog box.

d. Deleting a Drafter

To delete a drafter:

(1) From the Address menu, select Drafter List...

This opens the Drafter dialog box.

(2) Choose a drafter to be deleted by selecting one from the Name list.

(3) From the Drafter dialog box, select the Delete... button.

This deletes the drafter selected.

For Your Information: Editing and Deleting the Releaser Information works similarly.

4.10 TUTORIAL - PREPARING AND EDITING A MESSAGE

4.10.1 Overview

The functions described in this section are easier to understand if you keep in mind the basic steps needed to prepare a message, listed as follows:

a. Create a New Message or Open a Saved Message - to compose a new message or recall an existing message.
b. Edit Field Contents and Fill in Fields - to put required information into the appropriate fields of the sets, repeating fields as necessary.
c. Edit sets - to modify the structure of the message by repeating or inserting segments or sets until the set structure matches the desired message structure.
d. Check Spelling - Use the spelling checker to check the spelling (NOTE - This function not available at this time).
e. Errors in Message - Use the error routines to find any message errors.
f. Address the Message and Output it - Fill out the information on recipients, and output the message onto paper or disk.

4.10.2 Creating a New Message

The Message Generation Module allows you to use a message template to create a new message, which can then be tailored. In this example, you will use the MTF GENADMIN template to create a message.

a. To create a new message:

(1) There are two ways to create a new message: a) from the CMP Message Handler window, select NEW or NEW Button, or b) launch the CMP Message Generator directly from **CMP/JMPS/bin** using command **jmps <file name>**. (NOTE - The latter method is not for a DII user)

(a) You must fill out a "header" for the message first.

(2) After the message generator is launched, the Empty Message Input file window will be displayed. From the cascading menu, click on GENADMIN to select it. Press the open button to open new GENADMIN message template. (Double-clicking on GENADMIN also executes the command of creating a new GENADMIN message in the Empty Message Input File).

4.10.3 Editing Field Contents

Once in the message template, you can begin filling out the fields within each set. Fields are the basic building blocks of a message. They hold all the information contained in the message and provide format and identification information. To make a field active, click with your mouse in a field or TAB between fields.

Field Map and Range Character can always be found in the field status area of a message for the currently active field. Maps and ranges are a combination of letters shown in the example below. Lowercase letters are used to indicate the data format used in the field. The combination of characters shows what type of characters can be placed in each character position in the field.

For example, if the field map "**1-20ANBS**" is displayed in the field status area, the entry contains a range of **1** through **20 A**lphabetic, **N**umeric, **B**lank spaces, and **S**pecial characters. The table below explains each of the characters used in a field map.

Field Map Character	Meaning	Possible Characters
A	Alphabetic characters	(A-Z)
N	Numeric characters	(0-9)
B	Blank spaces	
S	Special characters	.:()-,?/ Note: the colon and forward slant are only allowed in the free text sets.
UND	Unlimited Text	a n b s
.	Decimal point	
C	Coded entry	(refer to the help text or Joint User's Handbook for more information)

For Your Information: If you try to enter a "/" in a non-free text set, you will hear a tone and the character will not be entered.

When editing, the BACKSPACE key, the left and right arrow keys or the mouse can be used to correct mistakes. The backspace key will erase the character to the left of the cursor and move the cursor one character to the left. The left and right arrow keys move the cursor from side to side. Text in a field can be selected with the mouse and commands under the Edit menu applied to the text.

After the field is completed, use the TAB key to move the cursor into the next field to the right, or click in it with your mouse using the left mouse button. If you are editing the last field in a set, the cursor will move to the first field of the next set if the TAB key is pressed. Pressing TAB while holding down the shift key will move you into the field before (i.e., to the left of) the field being edited.

As you tab or use the mouse to move out of a field, the program will validate the contents of the field. To validate the field, it compares what the field contains with the rules that define the format of the field. If the contents do not fit the rules, a field validation error will be displayed in the Error Warning field in the status area at the bottom of the message. The field can be corrected either at this point or at a later time. Fields with errors are highlighted background or field turns gray to distinguish them from correct fields.

a. Filling in fields

To fill in fields:

(1) Notice that the cursor is blinking in the first field in the EXER set. If the cursor is not in the first field, click on the first field with the left mouse button to select it.

The cursor's location is shown by the blinking cursor. The cursor indicates where the next characters will be inserted when you start typing.

(2) Type in the words **TextExercise**. Then press the TAB key.

The cursor is now in the next field. The program automatically displays the entered characters in all uppercase, which is a requirement of the MTF standard.

(3) Type in the words **MoreInformation**. Then press the TAB key.

The cursor's location is now in the first field of the next set, which is below the EXER field. Notice that the cursor is now blinking in this field.

(4) Press the TAB key while watching the bottom of the display.

The bottom status area displays information about the set you are in on the left and the field you are in on the right The status area is updated each time you move to a different set or field (see paragraph 4.8.1 for more information).

(5) Fill in the rest of the message to match the information in the figure below. The information in the following sections will aid you. Notice that you have filled in all of the mandatory fields that were indicated by a red box around the field.

b. Using Real Time Help ((NOTE: This function is not implemented at this time))

Real-time help provides help on the particular field, set, or message that is currently being used. The Real time help changes when moving from field to field, from set to set, and from message to message. Real time help is shown in a separate window that floats on top of all other windows. When real time help is selected from the Help menu in the Message Edit Window, a window is opened. This window updates in the same way the status area updates when moving from one field to another.

To turn on Real Time Help:

From the Help menu, select Real Time Help.

This opens the Real Time Help.

To turn on Field help:

(1) From the toolbar in the Real Time Help window, click on the Field button. This shows the field help in the Real Time Help window.

(2) Now click on one field in the message, then tab to another field.

Notice that the Field Help changed when you tabbed into another field, giving you help on the new field.

For Your Information: To get the other types of real time help, just press the appropriate button located across the top of the real time help window. Try pressing the Message, Set, Example, and Notes buttons to see what kind of information is provided for each of these.

To turn off Real Time Help:

(1) From the Help menu, select Real Time Help.

This is the same way you turned Real Time Help on. By selecting the Real Time Help again, the Real Time Help window is closed.

(2) For this exercise, turn the Real time help back on.

c. Clearing fields

To clear text in a field:

(1) Make the EXER set field containing "MOREINFO" active by clicking in the field.

(2) To clear the text, select Clear Field from the Edit menu. ((This function is not implemented at this time))

For Your Information: Larger objects such as sets can also be cleared by selecting them and choosing Clear Set.

d. "Undoing" an action ((This function is not implemented at this time))

Selecting Undo from the Edit menu will reverse the most recently executed action.

To undo an action:

From the Edit menu, select Undo Clear.

Notice that the text you just erased now reappears.

Selecting Undo a second time will redo the action that was just undone. Undo works for most commands and all message editing functions.

e. Pre-populated Fields

Some fields, as specified in the format for the message, have data in them before editing is begun. These fields are called pre-populated fields. The first field in the MSGID set is an example of a pre-populated field. These fields may not be edited.

f. Alternate Content Field

Many sets have fields whose contents may be formatted in two or more ways. Any field like this is called an alternate content field. The program provides easy access to alternate content fields. If a field is an alternate content field, then "Alt" will be displayed in the field status area at the bottom of the screen.

To toggle through the alternates:

(1) Use the scroll bar on the right to scroll down to the POC set.

(2) Click with the mouse into the third field of the POC set.

Notice that there is an "Alt" in the field status area, indicating that this is an alternate content field.

(3) From the View menu, select Next Field Alternate repeatedly until the alternate field is located. The current field type is displayed in the field status area.

To show all of the possible fields for an alternate contents field, select the Field Alternates cascading menu from the View menu. From the Field Alternates cascading menu, you can choose a field alternate directly.

Keyboard Method: The F2 key can also be used.

For Your Information: If information is entered into the default field alternate and it does not match the format of this alternate but does match a different alternate, when the message is saved and reopened, the system will select the best match for field alternate. If a field alternate is chosen, this field alternate will be displayed upon reopening a saved message.

g. Repeating Field Groups

All field groups (some may consist of only one field) can be repeated as necessary. Repeatable field groups are shown by an "R" flag in the field status area at the bottom of the message.

To repeat a field group (fg):

(1) Select the last field in the REF set by clicking on its field delimiter.

(2) From the Edit menu, select the cascading menu item Repeat Field Group. From the cascading menu Repeat FG, select 1.

This repeats the selected field group once. When a field group is repeated, an empty duplicate of the field is pasted immediately after the current selection. Field contents are not copied. To repeat the field group more than five times, select the menu item more... from the Repeat cascading menu. This will bring up a dialog box that allows the user to type in the number of times to repeat the field group.

(3) Select Undo Repeat FG from the Edit menu to undo the repeat. (NOTE: NOT AVAILABLE AT THIS TIME).

Keyboard Method: A short-cut for repeating a field group is to press the RETURN key after filling out a field group to repeat it. If duplicate fields are not required, hit return again and it will disappear.

For Your Information: To repeat a field group containing more than one field, the field group can be selected either by using the Field Group item in the Select menu or by selecting a single field in the field group, as described above.

h. Coded Fields

Coded field are fields that have a list of valid entries associated with them. The user can select from a list of possible entries by pressing on the pop-up menu button next to the field.

To edit coded fields:

(1) Fields that are coded can be edited by pressing the pop-up menu button to the right of a coded.

(2) A pop-up menu with the valid code entries for that field, according to Joint Pub 6-04 for USMTF messages or the service standard from which the message is defined, will appear. The proper code can be selected from this list.

For Your Information: Alternately, the user can type the desired value directly into the field.

4.10.4 Editing Sets

a. Inserting Sets

The MTF format allows the modification of messages by adding, deleting, and repeating sets or segments (groups of sets) in a message. The program supports the editing of sets through a variety of functions. Most of these functions are available from the Edit menu.

To insert a set:

(1) Select the REF set by clicking on its set id. The inserted set will go below the selected set.

A box border around the entire set shows that the set is selected.

(2) From the Edit menu, select menu item Insert Set.

When Insert Set is selected, a cascading menu pulls down. The choices on the menu are the sets that are valid to insert at this point.

(3) To insert at this point, the user would simply click on the set name from the cascading menu. To cancel at this point, click anywhere except on the cascading menu.

b. Repeating Existing Sets

Certain sets in a message are repeatable as required. By selecting the set and using the Repeat menu item from the Edit menu, the set can be repeated as many times as necessary. Repeatable sets are shown by an "R" flag following the set name in the status line. These are the only sets that are repeatable.

To repeat a set:

(1) Select the REF set by clicking on the set id.

A box border around the entire set shows that the set is selected.

(2) From the Edit menu, select menu item Repeat Set

Selecting Repeat pulls down a cascading menu. The choices on the menu are the number of times that the set will be repeated.

(3) From the cascading menu, choose 2.

Notice that the REF set is repeated one time, but the information it contained is not.

c. Repeating Rows in columnar set 5

Rows in columnar sets (noted by beginning with a number) may be repeated.

(1) Select the first row in the columnar set.

(2) From the Edit menu, select Repeat Row and select the number of rows from the cascade menu to be added.

4.10.5 Saving Messages

Users should save their work frequently. While computers are reliable for the most part, there are time when some work will be lost, such as during a power outage. Saving frequently will minimize the amount of work lost if the computer malfunctions. Once a message is created, the message should be saved. Saving a message will write to the hard disk so that it can be recalled and edited later.

a. To save a message:

(1) From the File menu of the Message Edit Window, select Save.

Many commands, including Save, can be accessed by simply selecting the toolbar button that corresponds to the command. This is an alternate way of selecting a command. The save command is the first icon on the Message Edit Window

4.10.6 Spelling Checker ((This function is not implemented at this time))

The spelling dialog box checks the message for spelling errors and suggests possible corrections.

a. To check the spelling of a message:

(1) From the Tools menu in the Message Edit Window, select Spelling...

This will open the spelling dialog box and begin searching the free text sets in the message for spelling errors. When the first error is found, the Unknown Work field will be populated with the first unknown word.

(2) The user has a number of choices.

"Ignore" should be used if the user wants to keep the work as it is currently spelled.

"Ignore All" is used if the user wants the spell checker to ignore all occurrences of the unknown work in the entire message.

"Change" is used if the work in the Change To text box is the correct spelling of the word, and the user would like the system to change the word to the correct spelling. The user may also enter an alternate spelling in the Change To field and use the Change button.

"Change All" is used if the user would like the spell checker to change all occurrences of the unknown word to the Change To word.

"Suggest" is used if the correct spelling is not in the Change To text field or the scrolling list. The user can type a spelling thought to be correct into the Change To text box, and then press Suggest to see if the word is in the spell checker.

"Suggestion" is a scrolling list that contains the other words close in spelling to the unknown word. If the correct spelling is contained in the list, users can select the correct spelling and press Change To to replace the unknown word with the selected word.

"Search" is a group of radio buttons that allows the user to specify the direction and scope of the spelling check. Up will cause the spelling checker to search up from the cursor position to the beginning of the message, Down will search down to the end of the message, and All will search the entire message.

(3) To close the Spelling dialog box, press Cancel.

4.10.7 Validating Messages and Error Correction

Validating a message will check the structure and the contents of the message against the appropriate message formatting rules to verify that it conforms to the message standard governing that message.

The program divides errors into two main types, field/set/message errors and structural errors. Field/set/message errors are errors that occur within the fields, sets, or message. Structural errors are errors concerning the structural notation of the message. However, both of these types of errors are handled under one “ERRORS” button.

a. To show field, set, and message errors

(1) From the Tools menu, choose "Errors". This will bring up a message box showing that the system is validating the message and asking the user to please wait. The Field/Set/Message Errors dialog box below will appear and the system will automatically locate the place in the message where the first error is located. The error will be highlighted in the message. Depending on the message window size, the user may need to scroll horizontally to see errors since the error may be off the screen.

"Error Type" displays the type of error (e.g., field error or set error).

"Description" displays a description of the current error.

"Set ID" displays the set id of the set containing the current error.

"Field Reference and Use No." displays the field reference and use number for an erroneous field, if the error type involves a field.

"Field Position" displays the position of the erroneous field in the set, if the error is a field error. If the error is not a field error, the text field is empty and the label grays out.

"Current Field Value" displays the current contents of the erroneous field; otherwise, it is empty and the label is grayed out.

"Change Field To:" allows modification of the contents of erroneous fields when field errors have occurred. To modify the contents, simply type the correct field data on this field and press the Change Field button. The System will validate the new field contents before changing the contents of the erroneous field in the message.

"Next Error" revalidates the field and displays a warning if the field is still invalid. It then updates the fields in the dialog box to display information about the next error found in the current message, and brings the cursor to the location of the next error in the message. When the end of the message is reached, a message box will indicate this, allowing users to end the error correction or continue to the beginning of the message.

"Prev Error" revalidates the field and displays a warning if the field is still invalid. Then it updates the dialog box fields to display information about the previous error in the current message and brings the user to the location of that error in the message.

"Change," for errors involving fields, updates the erroneous field's contents in the message with the contents of the Change Field To: text field after checking the validity of the new contents.

"Validate" validates the current message.

In the message window, the error location is indicated by a red box around the set in which the error is contained and the field is gray.

(2) At any time while correcting errors in a message, the user can click on the Message Edit Window to make it active and correct errors directly on the message using the message edit functions. In fact, for set level errors, this may be the only way to correct errors. Then, click on the Field/Set/Message Error dialog box and choose the Validate button.

(3) To close the Field/Set/Message Error dialog box, press Close.

(4) To redisplay the Field/Set/Message Error dialog box, select Show Field/Set/Message Errors from the Tool menu.

For Your Information: Fields with errors will have a gray background. The field status will display an "Err" message for these fields.

b. To show structural errors

Structural errors occur when a message's structural notation (SNV) rules are violated. These types of errors check for set conditionality violations, etc. For example, if both an EXER and an OPER set are in a message, an SNV error will occur.

(1) From the Tools menu, choose Errors...

This displays a list of all of the structural errors in the current message. The location of the error in the message is not presently indicated. Correct the errors directly in the Message Edit Window. After making changes, press the Update Errors button to update the list of remaining errors. Next Error and Prev Error move the user through the error list.

c. Correcting an Error in an Error Set ((This function is not implemented at this time))

When an error such as a missing set id occurs, the program needs help to correct the set. For this type of error, the erroneous information is put into an "error set", displayed as a red box with pink crosshatching. The entire set is contained in the error set. To correct the error, edit the erroneous part of the set (correct the set id, add a

missing end of set mark, etc.) when the field is tabbed out, the corrected set will revert to a normal set.

4.10.8 Previewing and Printing Messages

The preview command will show how the message will look when it is printed out or put in transmission ready form. While the message may be previewed in this state note that it cannot be edited.

a. To preview a message

(1) From the File menu, choose Preview. The dialog box below will be shown. Use the scroll bar to scroll through the message. Segments in the message are also displayed.

(2) When the message preview is finished, press Close to close the dialog box.

b. To print the open message

(1) From the File menu, choose Print. This will open the dialog box , giving you a number of options to select.

(2) To print with the print settings as they are, press the Print button, or press the Return key.

4.10.9 Filling in the Header (NOTE: For JMPS standalone only.)

In order to enter header data note that it is first necessary to create the message, save the message in the working directory, retrieve the message from the working directory, and then select the header format which one wishes to apply to the message.

A message can be saved with or without address information. All information pertaining to message addressing is contained in a message header.

a. To fill in a header

(1) From the Edit menu, choose the menu item Header.... This will cause the following dialog box to appear. The header format type must be chosen.

(2) Select the desired header format, and then press OK.

The message header dialog box below will appear. Many of the items have default values. The user only needs to change them if they are not the desired values.

(3) Keep the classification as **Unclassified**.

(4) The date and time fill in automatically. To update, press the button labeled Update DTG.

(5) The From: text box will be pre-populated with the default originator, if one was set. If not, or to change it, select the ¨ button. When the ¨ button is pressed, the Address Selection dialog box will appear.

From the Address Selection dialog box, the user can choose the address for the From: text box.

(6) To choose an address to be selected, click on the address in the Address scrolling list and select office symbols from the Office Symbols scrolling list, then press the Add button. This adds the address and the office symbols that were selected to the Addresses Selected scrolling list. When selected, an asterisk appears to the left of the office symbol in the Office Symbol scrolling list indicating that the office symbol has been added, and to the left of the long address.

To remove an address from the Addresses Selected List, select the address from the Addresses Selected List and press the Remove button.

To remove office symbols from an Address that is selected, select the address from the Addresses Selected List, then select the office symbols to be removed from the Office Symbols scrolling list and press the Remove button.

(7) Select OK to exit the dialog box.

(8) In the To: text box, type in DIR JITC. Addresses can also be selected as described above, using the button after the To: field.

(9) Press Save to Message and Close to save this information with the message and return to the Message Edit Window.

For a detailed description of all the header features, see "Creating a New DD173 or JANAP" in paragraph 4.12.5.

For Your Information: For the From, To, and Info fields, addresses can be selected if entered previously in your address list, or addresses can be typed directly into the field.

b. To edit an existing message header

(1) With the message open on the screen, choose Header... from the Edit menu. The header associated with this message will open.

(2) Perform any edit operations to the header, then select Save to Message and Close. Changes will be saved with the message.

c. To remove a message header from a message

(1) Open the message containing the header to be removed.

(2) From the Edit Menu, choose Remove Header.... This permanently disassociates the header information from the message.

4.10.10 Editing an Existing Message

Messages that have been created and saved can be open for further editing or reused as templates to create new messages.

a. Opening a Saved Message

Existing messages are opened from the Temporary Window.

To open an existing message:

(1) To get to the Temporary Window, select Temporary Window from the Windows menu.

The Windows menu is a list of all the windows that are open. This menu provides an easy way to jump back and forth between windows.

(2) Highlight the message called Trainer Message by clicking on it.

(3) From the toolbar, select Edit push button.

For Your Information: Another way of opening a message is to double-click on the message name in the Temporary Window, or to press the Open button in the Toolbar.

b. Columnar Sets

Many tactical messages contain columnar sets. A columnar set is a set whose fields are ordered arrangements of data aligned in columns (like numbers in a table).

Data is entered into columnar sets just as it is entered into linear sets, except that it is entered below the field label instead of to the right.

Repeating Rows in Columnar Sets:

A row in a columnar set can be repeated in two ways. One way is to select any field in a row (by clicking on a field delimiter) and selecting the Repeat FG command from the edit menu.

A second way is to use the Field Group command in the Select Menu to select the field group, then use the Repeat FG command in the Edit menu.

c. Segment Bars

Another View option is to show the segment bar icons. The segment bar icons are bars showing where segments are located within a message. Segments are groups of sets that can be repeated or edited. Segments may be nested. This option is only available in messages containing segments.

(1) To toggle on segment bars

From the View menu, select Show Segment Bars. The following is the result of toggling on the segment bars. The segment bar icons display to the left of the message body and act like buttons. A segment may be selected by clicking on its respective segment bar icon. Mandatory segments have red bars and optional segments have green bars.

(2) To save the message with the name entered, press the OK button. (NOTE: THIS FUNCTION IS NOT AVAILABLE AT THIS TIME).

4.10.11 Views

There are a number of ways that messages can be viewed in the Message Edit Window.

a. Field Map Characters . (NOTE: THIS FUNCTION IS NOT AVAILABLE AT THIS TIME).

A message can be displayed with the field map characters in the fields themselves (see "Editing Field Contents" in paragraph 4.10.3 for more information).

To toggle on field map characters;

From the View menu, select Show Field Maps.

This toggles on the field maps, changing the active fields in the message from empty fields to fields that have field maps.

(2) To select a segment

Click on the segment bar icon to the left of the segment desired:

The bar will look "pressed in" and a selection box will appear around the sets in the segment, indicating that the segment is selected. While the segment is selected, all edit operations will apply to the segment.

(3) To deselect a segment

Click on the same segment bar, click on a new segment bar, or click in any field.

4.10.12 Cutting, Copying, Deleting, and Pasting . (NOTE: THIS FUNCTION IS NOT AVAILABLE AT THIS TIME).

This program allows parts of messages to be copied and pasted to other parts of the message. It also allows parts of messages to be deleted. These actions should be performed in compliance with the MTF message or the rules governing the standard from which the message is defined; otherwise they could result in an invalid message.

"Cutting" deletes the item selected and places it into a buffer.

"Copying" creates a duplicate of the item selected and places it into a buffer.

"Deleting" erases the item currently selected.

"Pasting" inserts the contents of the buffer after the currently-selected object, or after the cursor in the case of text.

"Undo" will reverse the previous action. For example, a set may be deleted and a decision made to get it back. Select Undo Delete. The Undo command must be used immediately after the action for the action to be reversed.

a. To cut, copy, or delete a set

(1) Select the set by clicking on the set id or, from the Select menu, choose Set to select the set in which the cursor is located. This directs the program to work on this particular set.

(2) From the Edit menu, the user may select Cut, Copy, or Delete. For this example, select Copy Set. By selecting Copy, the user puts a copy of the selected field on the clipboard.

b. To paste the set

(1) Select the location to which the set should be pasted by clicking on a set in the message where the set is to be inserted.

(2) From the Edit menu, select Paste Set. Copied objects are pasted after the currently selected object. To paste a set after a second set, copying the first, select the second set after copying the first and select "paste" from the Edit menu. The command prompt in the status area reminds the user of the correct sequence of events.

c. To undo pasting the set

From the Edit menu, select Undo Paste Set.

This removes the pasted set. The Undo command must be used directly after the command to be "undone". At this point, the user cannot undo the pasted field. The user has to delete the pasted field to remove it.

d. To delete a set

(1) Select a set by clicking on its set id or choose Set from the Select menu.

(2) From the Edit menu, select Delete Set. This deletes the selected set.

(3) From the Edit menu, select Undo Delete.

NEVER DELETE THE LAST OCCURANCE OF A MANDATORY SET. The message will not validate if it does not contain all of the mandatory sets.

For Your Information: Cutting, copying, pasting, and deleting segments works in the same manner as the set operations. Segments are selected by clicking on segment bars.

4.10.13 Deleting and Appending Fields

Editing Fields works slightly differently from editing sets and segments. Fields cannot be cut, copies, or pasted. They can, however, be deleted and appended to sets.

a. To delete a field

(1) Select the field by clicking on its delimiter (/) or choose Field from the Select menu.

(2) From the Edit menu, select Delete.

b. To append a field to a set . (NOTE: THIS FUNCTION IS NOT AVAILABLE AT THIS TIME).

If a message contains a set or sets with missing fields, the fields can be restored using the Append Field command. Since fields will always be missing from the end of sets (because fields shift left when a set is deleted), Append Fields adds fields of the appropriate type to the end of sets. Once a set contains its full complement of fields, the Append Field command will be disabled for that set.

(1) Select the set id of the set with the deleted field.

(2) Set Append Field from the Edit menu.

The set is restored.

4.10.14 Restoring a Message . (NOTE: THIS FUNCTION IS NOT AVAILABLE AT THIS TIME).

The Restore Message command restores any fields, sets, or segments that were present in the original message template but are not in the current version of the message. This command does not affect data in the fields of the message.

a. To restore a message

From the Tools menu, select Restore Message.

4.10.15 SunSPARCstation

Mappings

L1	New Message	Review Message	L2
	Restart	Identify Set	
L3	Recall Msg	Print Msg	L4
	Compose Msg	Enter Argument	

L5	Convs Mode	Validate	L6
	Help	Scroll Alternate	
L7	Save Msg	Command	L8
	Clear Field	Repeat Field/Line	
L9	Clear Identify	MergeDat	L10
	Page Down	Page Up	

5 OTHER MESSAGE FUNCTIONS

5.1 SCOPE

In addition to inbound and outbound message processing, the Common Operating Environment (COE) Message Processor (MP) performs several other key functions. Normalization software converts message data into a format usable by host application software, e.g., Maneuver Control System (MCS). The Map Definition Language MAP (MDLMAP) program automatically generates messages in MTF format from information extracted from operational databases. The BOM to COM software converts BOMs to COMs and vice versa. The Journaling server provides an operational message journal for inbound and outbound messages, and allows information to be extracted from those messages. Finally, three other stand-alone tools are MTFVAL (for message validation), MTFXTRACT (to execute queries and extract data from a message) and MTFREPORT (to print a message in report format).

Thus, the remaining subparagraphs in this manual are as follows:

5.2 Normalization Software

5.3 Automatic Message Generation Using MDLMAP

5.4 Bit-Oriented Message (BOM) to Character-Oriented Message (COM) Software

5.5 Message Journaling Server

5.6 MTF Tools

5.2 NORMALIZATION SOFTWARE

5.2.1 Identification

This paragraph describes DII COE version 1.0/1.1, of the Normalization software CSCI for the Army Common Operating Environment (COE) Message Processor.

5.2.2 Purpose

The purpose of this paragraph is to describe the Normalization software CSCI, its functionality, the Application Programming Interface (API), and its rules for usage. This paragraph contains the following subsections: Referenced Documents; Execution Procedures; and Error Messages. Sample usage of the software and sample data files are included in the final subsections.

Referenced Documents

Document Number	**Title**
WP94B0000076	The Joint Message Analysis Processing System (JMAPS) User's Manual

5.3 AUTOMATIC MESSAGE GENERATION USING MDLMAP

5.3.1 Introduction

Different systems that must exchange information frequently do not represent this information in the same format. This causes interoperability problems, because the output from one system cannot be directly read as input by another. Instead, an operator has to manually convert one formal to another. This introduces errors and delays into the communications process.

The MDLMAP program described in this report is designed to solve some of these interoperability problems. MDLMAP works as a simple kind of interface "glue" for connecting systems together. The "source" system produces output text in its own format. MDLMAP transforms this text into the input format required by the "destination" system, using a "map file" describing the required transformations. MDLMAP was originally developed to generate USMTF messages from information extracted from operational databases. This "automatic message generation" capability is easier and faster than existing "manual" methods, which require an operator to query the database and then type or cut and paste the retrieved data into a message format. Typically, automatic message generation is only the first step in the message preparation process; a human operator edits the partial message, inserting information not available in the database, and/or reviewing the message before it is released. It is also possible to arrange a fully automatic message generation approach, where messages are created and transmitted without any operator intervention.

5.3.1.1 Text Formats.

In the context of this report, "formatted" text is text that is constrained to follow simple, rigid syntax rules. For example, the text of a MTF message consists of a sequence of records (called sets in the USMTF standard), each consisting of a sequence of fields, each of which is a text

string. Special strings are used to separate fields and terminate sets*. Figure 5-1 shows an example of a record and a field in a MTF message.

MSGID/ATOCONF/TACC/011/JUL// record

PERID/190600Z/59:20059Z// // = terminator

AIRTASK/UNIT TASKING//

TASKUNIT/59TFS// / = field separator

msndat/207/-/GLIDER 41/4F15/CAR / -/D10/-/33541//

Figure 5-1. Records and Fields in a USMTF Message

* The MTF standard imposes additional constraints on messages based upon the presence or absence of certain sets and upon the contents of certain fields.

MDLMAP is designed to process text that conforms to this basic, underlying format. The details may change with different formats; the records may be different, or contain different fields, but the basic format of the text must be a sequence of records of fields.

5.3.1.2 Text Format Translation.

MDLMAP translates text from one format to another by rearranging the input fields, perhaps deleting some fields and possibly adding some new text. For example, the output of a database report generator might appear as follows:

21392,	14095Z	A234,	SCOOP,	011,	435240N0751826W
21395	14105Z	A202,	DOG	011,	435240N0741820W

This text could be translated by MDLMAP into the following MTF message fragment:

MSGID/TACELINT//

SOI/21392/14095Z/14095Z/A234/SCOOP/011//

EMLOC/-/F/LS:435240N0751826W//

SOI/21395/14105Z/14105Z/A202/DOG/011//

EMLOC/-/F/LS:435240N0751820W//

The ability to translate the output format of database report generators into the format of USMTF or ADatP-3 messages is the primary purpose of MDLMAP.

5.3.1.3 An Overview of MDLMAP.

MDLMAP is a processor for programs written in a special-purpose language called Map Definition Language (MDL). A program in MDL specifies a mapping or translation between an input and an output text file. When MDLMAP is in run, it reads its MDL program from a *map file*. As MDLMAP executes the program, it reads the input text, then writes the translation of this text as its output.

A MDL program consists of two parts. The first part is an *input specification*, which defines the sequence of records and fields in the input text format. MDLMAP uses the input grammar to parse the input text. The result of this parse is then used to assign values to variables in the second part of the MDL program, which is an *output expression* defining the output to be produced. MDLMAP executes the output expression, producing a stream of text that is returned as the program's result.

5.3.1.4 Contents of this Section.

This section consists of the following four paragraphs:

a. Paragraph 5.3.1 is the introduction (i.e., this paragraph).
b. Paragraph 5.3.2 describes the MDL programming language in detail.
c. Paragraph 5.3.3 describes the command-level interface to MDLMAP
d. Paragraph 5.3.4 is a brief tutorial for writing MDL programs.

5.3.2 The Map Definition Language

5.3.2.1 Composition of an MDL Program.

This section describes the parts of an MDL program, beginning with the basic elements of the program, then showing how these form the major sections of the program. This section then describes the steps MDLMAP takes to execute the program.

5.3.2.1.1 Low Level Elements.

At the lowest level of detail, an MDL program is composed of a sequence of *numbers, strings, identifiers, reserved words, comments*, and other tokens. These are described below.

A number is a sequence of decimal digits, optionally preceded by a plus (+) or minus (-) sign. Numbers must be integers, decimal points and fractions are not allowed. For instance, these are valid numbers:

0 +1234 -567890

A string is a sequence of characters enclosed in a pair of single quotes (') or double quotes (") are also considered as valid strings. Some examples are:

" " 'abcd' "lmnop" '0'

The backslash (\) character is an escape character in a string. Instead of representing itself, it changes the meaning of the following character or characters. The escape sequences recognized by MDLMAP and their meanings are:

newline	NL (LF)	\n
horizontal tab	HT	\t
vertical tab	VT	\v
backspace	BS	\b
carriage return	CR	\r
form feed	FF	\f
alert	BEL	\a
backslash	\	\\
question mark	?	\?
single quote	'	\'
double quote	"	\"
octal number	*000*	*000*
hex number	*hhh*	\x*hhh*

Each escape sequence represents a single character in the string. Most escape sequences are portable between machines. However, the octal and hexidecimal escape sequences produce characters that are not portable between machines having different character sets. A sequence of octal or hexidecimal digits ends with the first character that is not an octal or hexidecimal digit. Octal numbers contain at most three digits; hexidecimal numbers contain any number of digits.

Escape sequences are used primarily to put newline, single quote, and double quote characters into strings. For example, escape sequences are essential in the following string:

'Joe said, "I can\'t live without MDLMAP!"\n'

It is an error for a string to contain an actual newline character; that is, to span more than one line. In the following example, the string on the left is illegal. When the last character on a line is a backslash, then the newline character is ignored. The string on the right is legal and contains the characters "this string is allowed."

'this string is 'this string is \

illegal' allowed'

An identifier is a name supplied by the programmer. Identifiers consist of a sequence of letters, digits, and the characters '$', '@', and '_'. Identifiers may not begin with a digit. These are valid identifiers:

abcdef A123xyZ $frog @foo_bar@

Uppercase and lowercase characters are distinct. The strings abc, Abc, and ABC form three different identifiers.

Reserved words are names defined by the MDL language. They cannot be used as identifiers by the programmer. These are the MDL reserved words:

filler	loop	rec	tail
for	output	separator	terminator
input	recno	tag	

Comments are remarks inserted by the programmer; they are ignored by MDLMAP. Any text that is not part of a string constant and is preceded by /* and followed by */ is a comment. These comments cannot be nested. For example:

/* This is a valid comment*/

/* However, /* this comment */ is illegal */

Comments may be placed at the end of a line. All text from the characters // to the end of the line is a comment. For example:

// This is a valid comment

// So is this; /* and // are part of the comment.

5.3.2.1.2 High-Level Elements**.**

There are two main parts to an MDL program. The first part of the program is the *input specification*, which describes the format of the input text. It consists of three sections:

a. The *lexical declarations* specify how the input text is to be divided into fields and records

b. The *record declarations* specify the composition of each record type in terms of the fields it may contain

c. The *input grammar* describes how to determine whether a particular sequence of input records is a valid instance of the input format.

The second part of an MDL program is the *output expression*, which specifies how to generate the output text, in the output format. This expression, when evaluated, returns a string as its result. This string is the output of the MDL program.

Figure 5-2 shows the parts of a sample MDL program used as an example throughout this section.

```
separator = '/';                                    lexical declarations

    terminator = '!';

    rec person 'PER'  =  last first  ;

rec addr   'ADDRESS' = street city state ;          record declarations

        rec phone  'PHONE' = home work ;
    rec sold  'SOLD' = date { thing } ;

    input =                                                  input grammar
    list { person addr phone sales { sold } } ;

    output =

list {                                                       output expression

        person.first '  ' person.last  '\n'
        addr.street '\n'
        addr.city ', ' addr.state '  ' addr.zip  '\n'
      };
```

Figure 5-2. A Sample MDL Program

5.3.2.1.3 Execution of a MDL Program**.**

When MDLMAP is run, it performs the following steps:

a. The MDL program in the map file is read and compiled. Certain errors in the MDL program can be detected at this point. If any are discovered, MDLMAP prints an error message and terminates.

b. The input text file is read and divided into fields and records according to the lexical declarations. As each record is identified, it is compared to the record declarations. Any input record which does not match one of the defined record types is discarded at this point.

c. The sequence of records in the input text is *parsed* according to the input grammar. This produces a data structure called a *parse tree* which shows how the input record

sequence can be derived from the input grammar. If the input record sequence does not correspond to the input grammar, MDLMAP prints an error message and terminates.

d. The output expression is evaluated. MDLMAP uses the input parse tree and the text of the input fields to determine the value of the references in the output expression. The result of the output expression is the output of the MDLMAP program. If errors occur while evaluating the output expression, MDLMAP prints an error message and terminates.

5.3.2.2 Lexical Declarations.

The lexical declarations in the map file specify the record terminator and field separator for the input text. The separator is the string that indicates the boundary between two fields. The terminator is the string that indicates the end of a record. These strings are specified in the map file as follows:

separator = string;	(for example: separator = '/';)
terminator = string;	(for example: terminator = '//';)

These declarations are optional and may appear in the opposite order. The default separator is the string '/'; the default terminator is the string '!'.

It is an error if either string is empty; that is, both strings must contain at least one character. It is an error if the separator and terminator are the same string. (However, one string may be a suffix or prefix of the other.)

As MDLMAP reads the input text, it looks for both the terminator and the separator strings. When the separator string is recognized, the current field ends and a new field in the current record begins. When the terminator string is recognized, the current record ends; if there are more characters in the input text, a new record begins. The separator and terminator strings are always discarded; neither string ever becomes part of a field.

MDLMAP gives special treatment to white space* characters at the beginning and end of an input field. Leading and trailing white space characters are discarded, except that SPACE characters before the first and after the last non-white space characters are retained. This has the effect of removing NEWLINE characters that are not embedded in a field.

As an example, suppose MDLMAP processes the following input text using the default separator and terminator. (In this text, SPACE characters are represented by a '.').

FIRST.FIELD/..SECOND.FIELD

../

THIRD.

FIELD

!

Given this text, MDLMAP would create one record containing three fields. The value of the three fields would be as follows:

field#1	'FIRST FIELD'
field#2	'..SECOND FIELD'
field#3	'THIRD \nFIELD'

* A white space character is a SPACE, TAB, RETURN, NEWLINE, FORMFEED, or vertical tab character.

Figure 5-3 contains a second example of how MDLMAP divides input text into fields and records. The top half of the figure shows the input text; the bottom half shows the record and field contents.

Each time MDLMAP finds the end of a record in the input text, it consults the record declarations to determine the type of the record and whether it has found a valid instance of this record type. This is the subject of the next section.

Input Text

```
PER/DOYLE/ARTHUR/CONAN!
PER/HOLMES/SHERLOCK!
ADDRESS/222B BAKER/NEW YORK/NY!
PHONE!
SOLD/02 MAY 1889/TELEPHONE!
SOLD/03 JUN 1889/HORSE/CARRIAGE!
PER/WIMSEY/PETER!
ADDRESS/TALLBOYS/-/UK!
PHONE//555-1010!
SOLD/14 MAY 1932/PORT WINE!
SOLD/18 MAY 1932/RARE BOOK!
```

Fields and Records

record number	*tag field*	*data field #1*	*data field #2*	*data field #3*
01	PER	DOYLE	ARTHUR	CONAN
02	PER	HOLMES	SHERLOCK	
03	ADDRESS	222B BAKER	NEW YORK	NY
04	PHONE			
05	SOLD	02 MAY 1889	TELEPHONE	

06	SOLD	03 JUN 1889	HORSE	CARRIAGE
07	PER	WIMSEY	PETER	
08	ADDRESS	TALLBOYS	-	UK
09	PHONE		555-1010	
10	SOLD	14 MAY 1932	PORT WINE	
11	SOLD	18 MAY 1932	RARE BOOK	

Figure 5-3. Input Text Divided Into Fields and Records

5.3.2.3 Record Declarations.

Each record declaration in the map file specifies three things. It shows how to recognize an instance of this record type in the input text. It defines the number of fields that may appear in a record of this type. Finally, it supplies identifiers that refer to the record and its fields in the output expression.

Record declarations have the following form:

rec name tagstring = fixedfields { tailfields } ;

For example, here are the four record declarations from the sample map file in Figure 5-2:

```
rec person 'PER'          = last first ;
rec addr   'ADDRESS'      = street city state ;
rec phone  'PHONE'        = home work ;
rec sold   'SOLD'         = date { thing } ;
```

The record *name* is the identifier used in the output expression to refer to an instance of this record type. A record name has global scope and cannot be used as the name of a different record or in the input grammar as the name of a *repetition*.

The first field in each input record is the *tag* field; all subsequent fields are *data* fields. The type of the input record is determined by matching the tag field against the *tagstring* part of a record declaration. (If an input record's tag does not match any record declaration, MDLMAP prints a warning message and discards that record.) It is an error if two record declarations have the same *tagstring*.

If there is more than one record declaration in an MDL program, then each must have a *tagstring* part. (Otherwise there would be no way to determine the type of the input records.) If there is exactly one record declaration in an MDL program, then the *tagstring* part of the record declaration is optional. If the *tagstring* is omitted, then the input records do not have a tag field (that is, the first is also the first data field) and every input record is of the same type.

The *fixedfields* part is a list of identifiers used in the output expression to refer to the fields of a record. The first identifier in the list refers to the first data field in the record, and so forth. The reserved word filler may be used in the *fixedfields* part to denote a field that will be recognized in the input text but that cannot be referenced in the output expression.

The *tailfields* part declares a group of fields that may be repeated at the end of an input record. The repeated fields are known as the record's *tail repetition*. The *tailfield* identifiers are used in the output expression to refer to the repeated fields: for each element in the tail repetition, the first identifier refers to the first data field, and so forth.

The *tailfields* part of a record declaration is optional. When it is omitted we have a *fixed-length* record type; when it is included we have a *variable-length* record type. A missing field in the input text results in an empty field in the input record. If there are too many fields in the input text for a fixed-length record, MDLMAP prints a warning message and discards that record. For example, given these record declarations:

```
rec fix 'F'  = f1 f2;                 fixed-length; exactly two fields
rec var 'V'  = v1 { t1 t2};           variable-length; one or more fields
```

MDLMAP handles this input text as follows:

```
F!              f1 is ' '; f2 is ' '
F/aa/bb!        f1 is 'aa'; f2 is 'bb'
F/aa/bb/cc!     invalid record; too many fields

V/dd!           v1 is 'dd'; the tail repetition is empty
V/dd/ee/ff/gg!  v1 is 'dd'; the tail repetition contains two elements:
                in the first tail element, t1 is 'ee'; t2 is 'ff';
                in the second tail element, t1 is 'gg'; t2 is ''.
```

Figure 5-4 depicts the sample input text from Figure 5-3 after MDLMAP has identified the types of the input records.

rec #	*record type*	*data field #1*	*data field #2*	*data field #3*	
01	*invalid*	DOYLE	ARTHUR	CONAN	*discarded: invalid # of record*
02	*person*	HOLMES	SHERLOCK		
03	*addr*	222B BAKER	NEW YORK	NY	
04	*phone*				
05	*sold*	02 MAY 1889	TELEPHONE		
06	*sold*	03 JUN 1889	HORSE	CARRIAGE	
07	*person*	WIMSEY	PETER		
08	*addr*	TALLBOYS	-	UK	
09	*phone*		555-1010		

10	*sold*	14 MAY 1932	PORT WINE
11	*sold*	18 MAY 1932	RARE BOOK

Figure 5-4. Input Text After Record Identification

5.3.2.4 Input Grammar.

The input grammar in a map file describes all of the permissible sequences of input records. It does this by declaring *groups* of records and *repetitions* of these groups. MDLMAP parses the input record sequence into instances of these groups and repetitions. The resulting data structure is used to determine the value of references in the output expression.

A group declaration is an ordered list of its members. Any record declared in the map file may be a group member. A group member may also be a repetition. (Repetitions will be described later in this section.)

The input grammar as a whole is a group; it has the following form:

input = *group* ;

For example, given the record declarations in the sample map file in Figure 5-2, we could write the following input grammar:

input = person addr phone ;

This input grammar states that the input text must consist of exactly one group, which must contain one **person** record, one **addr** record, and one **phone** record, in that order. In general, a group declaration is a (possibly empty) sequence of record types and/or repetitions. When MDLMAP attempts to find an instance of a group type in the input record sequence, it must find each member of the group in the specified order.

The declaration of a *repetition* always appears as one of the members of a group. The declaration has the following form:

name [maxrep] {group } ;

The repetition *name* is the identifier used to refer to this repetition in the output expression. It may also be used as a member of a subsequent group declaration in the input grammar. Repetition names have global scope. It is an error for a repetition to have the same name as a record type. It is an error to declare two repetitions with the same name.

The *group* part of the repetition declaration is the sequence of record types and/or repetitions in the current repetition. Since the members of a group may be repetitions, one repetition may be nested inside another.

The *maxrep* part is an optional upper limit on the number of times that the group may be repeated in the input text. If omitted, there is no limit on the number of elements. There is never a lower limit on the number of elements.

When MDLMAP finds an instance of a repetition in the input record sequence, it attempts to match the repetition's group as many times as possible, up to the limit imposed by *maxrep*. Each time the group is matched, one element is added to the repetition. MDLMAP always succeeds in matching a repetition to the input sequence, since every repetition is allowed to have zero elements.

As an example, we might write the following input grammar:

```
input = rep { person addr phone } ;
```

This grammar states that input text consists of a group containing one member, a **rep** repetition. Each element of the repetition must contain a **person**, **addr**, and **phone** record, in that order. The input text may contain any number of these elements.

To place an upper limit on the number of repetitions, we could write:

```
input = rep [10] { person addr phone } ;
```

This would permit a maximum of 10 groups to appear in the input text. (There would be no minimum number; the input text could be completely empty.)

One repetition may be nested inside another. This is done in the sample map file in Figure 5-2, where the input grammar is:

```
input = list {person addr phone sales { sold } } ;
```

This grammar states that the input text is a group with one member, a **list** repetition. Each element of this repetition is a group containing a **person**, **addr**, and **phone** record, followed by a **sale** repetition. A **sales** repetition is declared to consist of zero or more **sold** records. Figure 5-5 represents the parse tree created by MDLMAP using this grammar on the input record sequence from Figure 5-4.

Once we have declared a repetition type, we may refer to it by name in the remainder of the input grammar. For example, given record types A and B, we could write this grammar:

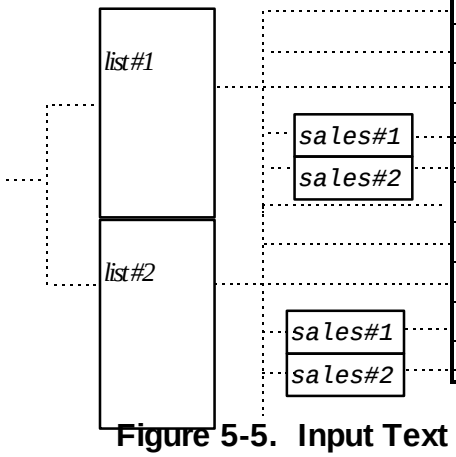
```
input = rp2 { A } B rp2;
```

This grammar states that the input text contains a number of A records, followed by exactly one B record, followed by a (possibly different) number of A records.

Suppose we give the above input grammar to MDLMAP, together with the following sequence of records: A, A, B, A. As MDLMAP parse the input record sequence, it recognizes the first two A

records as the first rp2; it next finds the required B record; it finally recognizes the final A record as the final rp2.

If we give MDLMAP the same input grammar and the input sequence A, A, then it will print an error message and terminate. MDLMAP recognizes the two A records as the first rp2 member of the input group. It then fails to find the required B record and consequently fails to parse the input text. When this happens, MDLMAP prints an error message and terminates.



rec #	record type	data field #1	data field #2	data field #3
01	invalid	DOYLE	ARTHUR	CONAN
02	person	HOLMES	SHERLOCK	
03	addr	222B BAKER	NEW YORK	NY
04	phone			
05	sold	02 MAY 1889	TELEPHONE	
06	sold	03 JUN 1889	HORSE	CARRIAGE
07	person	WIMSEY	PETER	
08	addr	TALLBOYS	-	UK
09	phone		555-1010	
10	sold	14 MAY 1932	PORT WINE	
11	sold	18 MAY 1932	RARE BOOK	

Figure 5-5. Input Text After Parser

```
INPUT PARSE ACTION REPORT:

attempting to parse a Group [input]
   parsing Rep (rp2)
   attempting to parse f Group (rp2)

   parsing a Record (a):OK
   Group (rp2) succeeded
   parsed Rep (rp2) #1
   attempting to parse a Group (rp2)

   parsing a Record(a):OK
   Group (rp2) succeeded
   parsed Rep (rp2) #2
   attempting to parse a Group (rp2)

   parsing a Record(a):FAIL (not a b)

   Group(rp2) failed
  done parsing Rep (rp2)
  parsing a Record (b):OK

  parsing Rep (rp2)


parsing a Record(a):OK
parsed Rep (rp2) #1
attempting to parse a Group (rp2)

Group (rp2) failed
done parsing Rep (rp2)
Group (input) succeeded
```

Parsing A, A, B, A (successful)

```
INPUT PARSE ACTION REPORT:

attempting to parse a
Group [input]
parsing Rep (rp2)
attempting to parse a
Group (rp2)
parsing a Record (a):OK
Group (rp2) succeeded
parsed Rep (rp2) #1
attempting to parse a
Group (rp2)
parsing a Record(a):OK
Group (rp2) succeeded
parsed Rep (rp2) #2
attempting to parse a
Group (rp2)
parsing a Record(a):FAIL
(EOF)
 Group (rp2) failed
done parsing Rep (rp2)
parsing a Record(b):FAIL
 (EOF)
 Group (input) failed
attempting to parse a Group
(rp2)
Group (rp2) succeeded


parsing a Record (a):FAIL (EOF)
```

Parsing A, A (not successful)

Figure 5-6. Parsing Action Reports from MDLMAP

MDLMAP is capable of printing a report of its action as it parses the input record sequence. Figure 5-6 displays the output text produced by MDLMAP for the two input sequences above.

It is possible to write an ambiguous input grammar in MDL. In this case, there will be some input sequences with more than one valid parse. For example, the following grammar is ambiguous:

```
input = rp3  { A } rp3  ;
```

Given an input sequence containing a single A record, there are two possible parses:

1. The first rp3 could contain one element and the second rp3 could be empty.

2. The first rp3 could be empty and the second rp3 could contain one element.

MDLMAP deals with ambiguity in the following fashion: when it parses a repetition, it always matches as many elements as possible. Consequently, in the above example MDLMAP will always produce parse #1.

Because of this rule, MDLMAP will sometimes report that its input sequence cannot be parsed even though a legal parse does exist. For example, the following grammar is both legal and unambiguous; however, using it, MDLMAP will always fail to parse the input sequence.

input = rp4 { A } A ;

The trouble is that no matter how many A records appear in the input sequence, MDLMAP will match all of them as it parses the rp4 member. There will never be an A record "left over" to match the final member in the input group. Consequently, MDLMAP can never succeed using this grammar. Users must be careful to avoid writing such a grammar.

5.3.2.5 Output Expressions.

MDLMAP first reads and parses the input text file according to the declarations in the map file. It then evaluates the map file's *output expression*. The string result returned by this expression is the output of the MDLMAP program.

Expressions are composed of constants, references, and operators. A constant is a string or a number, as defined in section 5.3.2.1.1. A reference is a name for part of the processed input text. An operator takes one or more expressions as its operands and returns some computed value.

This section describes how expressions in MDL are composed and explains how these expressions are evaluated to produce a result.

5.3.2.5.1 Printing the Parse Tree**.**

The reserved word input, when used by itself as the output expression, causes MDLMAP to print the parsed representation of the input text. This shows how the input text was broken into records and fields, and also shows how the input record sequence was broken down into repetitions and groups. This can be useful for debugging the input portion of the map file.

For example, given the map file from Figure 5-2 and the sample input text from Figure 5-3, MDLMAP produces the following output:

```
ValGroup (input)=[
ValRep (2*list)=[
01: ValGroup (list)=[
ValRecord(#1,person)=[HOLMES/SHERLOCK]
ValRecord(#2,addr)=[222B BAKER/NEW YORK/NY/10010]
ValRecord(#3,phone)=[]
```

```
ValRep(2*sales)=[
01: ValGroup(sales)=[
ValRecord(#4,sold)=[02 MAY 1889/TELEPHONE]]
02: ValGroup(sales)=[
ValRecord(#5,sold)=[03 JUN 1889/HORSE/CARRIAGE]]]]
02: ValGroup(list)=[
ValRecord(#6,person)=[WIMSEY/PETER]
ValRecord(#7,addr)=[TALLBOYS/-/UK/]
ValRecord(#8,phone)=[/555-1010]
ValRep(2*sales)=[
01: ValGroup(sales)=[
ValRecord(#9,sold)=[14 MAY 1932/PORT WINE]]
02: ValGroup(sales)=[
ValRecord(#10,sold)=[18 MAY 1932/RARE BOOK]]]]]]
```

This output can be understood as follows:

a. At the top level, the entire input is a group. This group contains one member, which is a list repetition.

b. The list repetition contains two elements, each of which is a group of four members: person, addr, and phone records, and a sales repetition.

c. Each sales repetition happens to contain two elements. Each element is a group having one member, a sold record.

5.3.2.5.2 References and Values.

A *reference* is an identifier used in the output expression that denotes some part of the input text. The value of a reference is the corresponding part of the input parse tree.

A reference is almost always one of the identifiers used in the input specification. (A reference can also be a user-defined variable; see section 5.3.2.5.3.16. For example, given the map file from the previous section, the identifier list is a reference to the list repetition that is the sole member of the top-level group.

The value denoted by a reference depends on the scope in which it appears. The scope of the output expression as a whole is the global scope, in which the only defined references are the members of the top-level input group. However, certain operators may introduce new, local scopes, in which other references are defined. (See sections 5.3.2.5.3.8 and 5.3.2.5.3.9).

A value will always be one of the following categories:

a. A value can be a *string*. A reference that denotes a specific input field returns a string as its value.

b. A value can be a *repetition*. A reference that is the name of a repetition defined in the input grammar returns a specific repetition as its value. A reference to the tail group of a variable-length record also returns a repetition as its value.

c. A value can be a *group*. A reference to a specific element of a repetition returns a group as its value. The fields of an input record also form a group.

d. A value can be *empty*. When a string is required, the **empty** value is the empty string "". When a repetition or group is required, the *empty* value has the property that any operator applied to it always returns the *empty* value.

e. A value can be *undefined*. When a string is required, the *undefined* value is the string "***UNDEFINED***". When a repetition or group is required, the *undefined* value has the property that any operator applied to it always returns the *undefined* value.

There is no distinct numeric type. As far as MDLMAP is concerned, a number is just a string that contains only digit characters (and perhaps an initial + or - sign).

5.3.2.5.3 Operators**.**

This section describes the operators in an MDL expression. The table below lists all of the MDL operators. The examples in this table use the following symbols:

expr	any expression
rep	an expression returning a repetition value
group	an expression returning a record or group value
rec	an expression returning a record
string	an expression return a string value

()	function call		*expr (expr_list)*
[]	subscripting		*rep [expr]*
[]	subrange selection		*rep [expr .. expr]*
.	member selection		*group.member*
.recno	record number		*rec* .recno
.tag	record tag string		*rec* .tag
.tail	record tail repetition		*rec* .tail
{}	qualification		*group { expr }*
{}	repetition		*rep { expr }*
[>>]{}	indexed repetition		for *[expr>>expr] {expr}*
		or	*rep [expr>>expr] {expr}*
loop	iteration		*loop [expr] {expr}*
#	length		*# rep*
!	logical NOT		*! expr*
*	multiplication		*expr * expr*
/	division		*expr / expr*
+	addition		*expr + expr*
-	subtraction		*expr - expr*
&&	logical AND		*expr && expr*
||	logical OR		*expr || expr*
==	string equality		*expr == expr*
!=	string inequality		*expr != expr*
>	string greater than		*expr > expr*
>=	string greater than or equal to		*expr >= expr*
<	string less than		*expr < expr*
<=	string less than or equal to		*expr <= expr*
==	numeric equality		*expr #== expr*
!=	numeric inequality		*expr #!= expr*
>	numeric greater than		*expr #> expr*
>=	numeric greater than or equal to		*expr #>= expr*
<	numeric less than		*expr #< expr*
<=	numeric less than or equal to		*expr #<= expr*
=	assignment		*variable = expr*
? :	conditional evaluation		*expr ? expr : expr*
none	concatenation		*expr expr*

Figure 5-7. MDL Operators and Their Precedence

a higher precedence than operators in all following boxes. Unary operators are right-associated; all others are left-associative. Parentheses override precedence and association. For example:

2*3+4	is 10; * takes precedence over +
2*(3+4)	is 14; the parentheses override operator precedence

3-2-1	is 0; the operators group left-to-right
3-(2-1)	is 2; the parentheses override operator association

In general, MDLMAP makes no assumptions about the order of evaluation. The conditional operator ?:, and the boolean operators && and || are exceptions; these are described in detail in section 5.3.2.5.3.14.

In the following operator descriptions, when we say that an operand "must be" a certain type, we mean that MDLMAP writes an error message and quits if the condition is not true. When possible, these errors are detected as the map file is parsed. However, some errors, especially those where an operand "must be" a number, cannot be detected until the output expression is evaluated.

5.3.2.5.3.1 Concatenation.

An expression of the form *exp1 exp2* is a concatenation expression. Both expressions must be strings. The result is the concatenation of the strings.

Example: assume str1 is "abc", and str2 is "def":

str1 str2 is "abcdef"

5.3.2.5.3.2 Subscripting.

An expression of the form *expr [index]* is a subscript expression. The first expression must be a repetition or string. The second expression (the *index*) must be a number.

When applied to a repetition, the result is a single repetition element determined by the index number: if 1, the first element; if 2, the second element; etc. When applied to a string, the result is a new string containing a single character determined by the index number.

If the index number is less than one, the result is *undefined*. If the index number is greater than the number of elements in the repetition or string, the result is *empty*.

Examples: assume list is the top-level repetition in Figure 5-5, and that str is "abcd":

list[1]	is the group containing the first five valid input records
str[0]	is *undefined*
str[1]	is "a"
str[4]	is "d"
str[5]	is *empty*

5.3.2.5.3.3 Subrange Selection.

An expression of the form *exp [first..last]* is a subrange selection expression. The first expression must be a repetition or string. The second and third expressions (*first* and *last*) must be numbers.

When applied to a repetition, the result is a new repetition containing a sub-sequence of the elements in *exp*, where *first* specifies the first element and last the last element of the sub-sequence. When applied to a string, the result is a new string containing a substring of *exp* specified by *first* and *last*.

If *first* is less than one, the result is *undefined*. If *first* is greater than the number of elements in the repetition or string, or if *first* is less than *last*, then the result is *empty*. If last is greater than the number of elements, then the last element in the result is the last element in *exp*.

Examples: assume that str is "abcd":

str[1..2] is "ab"	str[0..2] is *undefined*
str[2..4] is "bcd"	str[5..9] is *empty*
str[3..3] is "c"	str[3..2] is *empty*
str[3..9] is "cd"	str[3..0] is *empty*

5.3.2.5.3.4 ***Member Selection.***

An expression of the form *exp.member* is a member selection expression. The first expression must be a group or record. The second expression must be the identifier for a member of the group or record. The result is the specified member.

Examples: assume list is the top-level repetition form Figure 5-5.

list[1].addr	is the first addr record (that is, grp)
list[1].addr.street	is "222B BAKER"

It is possible for a named member to occur more than once in a group or record. For example, the following input grammar defines a group with two foo members:

```
input = rp5 {  foo  bar   foo  };
```

An expression of the form *exp.number%member* is also a member selection expression. The *number* must be a numeric constant. It specifies the occurrence of the named member. For example:

rp5[1].1%foo	(1st foo member of 1st rp5 element)
rp5[1].2%foo	(2d foo member of 1st rp5 element)

It is an error if the named member does not occur at least *number* times in the group or record.

5.3.2.5.3.5 Record Number.

An expression of the form *rec*.recno is a record number expression. The first expression must be a record. The result is the ordinal number of that record in the input record stream. Input records, which are discarded by MDLMAP because they do not match any record type or because they have too many fields, are counted for the purpose of determining each record's number.

Examples: assume list is the top-level repetition from Figure 5-5.

```
list[1].person.recno    is 2
list[1].addr.recno              is 3
list[2].person.recno    is 7
```

5.3.2.5.3.6 Record Tag String.

An expression of the form *rec*.tag is a record tag expression. The first expression must be a record. The result is the tag string associated with the input record *rec*.

Examples: assume list is the top-level repetition from Figure 5-5:

```
list[1].person.tag              is "PER"
list[1].addr.tag        is "ADDRESS"
list[2].person.tag              is "PER"
```

5.3.2.5.3.7 Record Tail Repetition.

An expression of the form *rec*.tail is a record tail repetition expression. The first expression must be a record. The result is the tail repetition of the designated record. This value can then be treated as any other repetition value.

If *rec* is a fixed-length record, or if its tail repetition contains no elements, then *rec*.tail returns the *empty* value.

Examples: assume rec1 is the second sold record appearing in Figure 5-5:

```
rec1.date is "02 MAY 1889"
rec1.tail  is the tail repetition, containing two elements
rec1.tail[1] is the first element of the tail repetition

rec1.tail[1].thing is "HORSE"
rec1.tail[2].thing is "CARRIAGE"
```

5.3.2.5.3.8 Qualification.

An expression of the form *group { exp2 }* is a qualification expression. The first expression must be a group or record. The result is the value of the second expression, which is evaluated within the local scope of *group*. That is, the members of *group* are defined as references within *exp2*.

Examples: assume that grp is the first addr record in Figure 5-5:

grp { street }	is "222B BAKER"
grp { city }	is "NEW YORK"

The first expression is evaluated once to establish the scope for the entire evaluation of the second expression. For example, assume rep[i] is the first addr record in Figure 5-5. Then, in this expression,

rep[i] {expr}

the value of street in *expr* is always "222B BAKER", even if the evaluation of *expr* changes the value of i as a side-effect.

It is possible that a reference defined in the global scope will become inaccessible in the local scope. For example, consider the following map file:

```
rec type = foo bar;
input = foo { rtype };
output = foo[1] { foo };
```

Within the qualified expression, the identifier foo refers to a record field, and not to the foo repetition in the top-level input group. The local definition of foo is said to shadow the more global definition, making it inaccessible within the local scope.

5.3.2.5.3.9 ***Repetition.***

An expression of the form *rep { loopexp }* is a repetition expression. The first expression must be a repetition. The second expression is evaluated in the scope of the elements of *rep*, once per element. The resulting values are concatenated to form the result of the overall expression.

Example: assume that list is the top-level repetition in Figure 5-5:

list { addr.street "$" } is "222B BAKER$TALLBOYS$"

5.3.2.5.3.10 Indexed Repetition.

An expression of the form for [*counter : from >> to] { loopexp* } is an indexed repetition expression. The *counter* must be an identifier. The *from* and *to* expressions must be numbers.

The third expression is evaluated zero or more times, depending on the values of *from* and *to*; the resulting values are concatenated to form the result of the entire expression.

When an expression of this type is evaluated, the *from* and *to* expressions are evaluated and the resulting values saved. Then, the *counter* variable is initialized to the value of the *from* expression. Finally, as long as the value of the *counter* is less than or equal to the saved value of the *to* expression, the *loopexp* expression is evaluated and the *counter* variable incremented by one.

The *counter* variable is defined only within the scope of the *loopexp* expression. It is an error to assign a new value to the *counter* variable within loopexp (see section 5.3.2.5.3.16).

The direction of the loop counter can be reversed; that is, the counter can be decremented with each iteration. This has the form for [*counter:to<<from]{ loopexp }*. The *counter* variable is still initialized to the value of the *from* expression. The *loopexp* expression is evaluated as long as the value of *counter* is greater than or equal to the saved value of the *to* expression.

The counter variable can be omitted, giving the form for *[from >> to] { loopexp }* or for *[to<<from]{loopexp}*. MDLMAP performs the same number of iterations; there simply is no *counter* variable to use as a reference within the *loopexp*.

Examples: assume that list is the top-level repetition in Figure 5-5:

```
for [i:1>>5] {i ","}     is "1,2,3,4,5,"
for [i:6>>5] {i ","}     is ""
for [i:1<<3] {i ","}     is "3,2,1,"
for [1>>5]  {"0,"}       is "0,0,0,0,0,"
```

```
for [i:1>>3] {i ":" list [i].per.last ","}

is "1:HOLMES,2:WIMSEY,3:,"
```

An alternate form of indexed repetition specifies a repetition instead of the for keyword; this has the form *rep [counter:from>>to] { expr* }. The number of iterations is determined by the *from* and *to* expressions, as before, but the *loopexp* expression is evaluated in the context of a particular element of *rep:* when *counter* is 1, the first element, etc. When using this form, it is an error for the *counter* variable to be less than 1. If *counter* is greater than the number of elements in the repetition, then *loopexp* is evaluated in the context of the *empty* value.

Using the alternate form and specifying a repetition, it is possible to omit the *from* expression, the *to* expression, or both. If the *from* expression is omitted, MDLMAP uses the value 1. If the *to* expression is omitted, MDLMAP uses the length of the repetition.

Examples: assume that list is the top-level repetition in Figure 5-5. Then,

```
list [i:1>>3] {i ":" per.last "," }
```

is "1:HOLMES,2:WIMSEY,3:,"

list [i:] {i: ":" per.last ","}

is "1:HOLMES,2:WIMSEY,"

5.3.2.5.3.11 Length.

An expression of the form *#exp* is a length expression. The sub-expression must be a repetition or string. The result is the number of elements in the repetition or the number of characters in the string.

Examples: assume list is the top-level repetition in Figure 5-5.

#list	is 2
#"abcde"	is 5
#""	is 0

5.3.2.5.3.12 Logical NOT.

An expression of the form *!expr* is a logical NOT expression. The sub-expression must be a number. The result is 1 (representing the *true* value) if the sub-expression has a the value zero; the result is 0 (representing the *false* value) if the sub-expression is nonzero.

5.3.2.5.3.13 Arithmetic Operators.

An expression of the form *exp1 op exp2*, where *op* is one of the four arithmetic operators +, -, *, or /, is an arithmetic expression. The two expressions must be numbers. The result is obtained by applying the specified operation to the expression values:

- \+ is addition
- \- is subtraction
- * is multiplication
- / is division; the result is the integer quotient

The most-positive and most-negative number accepted by MDLMAP is machine-dependent. It is an error to exceed these limits. It is also an error if the second operand of the division operator is zero.

5.3.2.5.3.14 Logical AND, Logical OR.

An expression of the form *exp1 && exp2* is a logical AND expression. The first expression must be a number. If *exp1* is zero, then the result is zero. If *exp1* is not zero, then *exp2* is evaluated (and must return a number); the result is zero if *exp2* is zero, and one otherwise.

An expression of the form *exp1 || exp2* is a logical OR expression. The first expression must be a number. If *exp1* is one, then the result is one. If *exp1* is not one, then *exp2* is evaluated (and must return a number); the result is one if *exp2* is one, and zero otherwise.

These expressions are two of the MDL expressions which guarantee an order of evaluation and which do not always evaluate all of their sub-expressions.

5.3.2.5.3.15 Comparison Operators.

An expression of the form *exp1 op exp2*, where *op* is one of the following six operators, is a string comparison expression. The two expressions must be strings. The result is one if the comparison condition is true, and zero otherwise. The string comparison operators are:

==	*exp1* and *exp2* are the same
!=	*exp1* and *exp2* are not the same
>	*exp1* is greater than *exp2*
>=	*exp1* is greater than or equal to *exp2*
<	*exp1* is less than *exp2*
<=	*exp1* is less than or equal to *exp2*

String comparisons are made using the character set of the processor running the MDLMAP program. This is almost always (but not necessarily) the ASCII character set.

A numeric comparison expression uses one of the following six operators. For these operators, the two expressions must be numbers.

#==	*exp1* and *exp2* are the same
#!=	*exp1* and *exp2* are not the same
#>	*exp1* is greater than *exp2*
#>=	*exp1* is greater than or equal to *exp2*
#<	*exp1* is less than *exp2*
#<=	*exp1* is less than or equal to *exp2*

Because MDLMAP will always convert numbers to strings as needed, the string comparison operators can sometimes yield surprising results. For example:

2 #< 10	is 1, because the number 2 is less than the number 10.
"2" < "10"	is 0, because the first character of the string "2" is greater than the first character of the string "10".
2 < 10	is 0, because the numbers are converted into strings, then compared.

It is good practice to use the numeric comparison operators whenever you wish to compare numbers.

5.3.2.5.3.16 Assignment.

An expression of the form *identifier = expr* is an assignment expression. The expression must be a number. The result of the expression is the empty string; however, as a side effect the identifier is bound to a new variable that contains the value of *expr*. Subsequent references to *identifier* within the current scope return the stored value.

Examples: assume list is the top-level repetition from Figure 5-5.

i=1 list [i].addr.street	is "222B BAKER"
i=0 i=i+1 list[i].addr.street	is "221B BAKER"

5.3.2.5.3.17 Conditional Evaluation.

An expression of the form *exp1 ? exp2 : exp3* is a conditional evaluation expression. The first expression must be a number. If *exp1* is not zero, then *exp2* is evaluated and its value returned as the value of the entire expression. Otherwise, *exp3* is evaluated and its value returned.

This is one of the MDL expressions that guarantees an evaluation order and does not always evaluate all of its sub-expressions.

Examples:

0 < 2 ? "yes" : "no"	is "yes"
0 > 2 ? "yes" : "no"	is "no"

5.3.2.5.3.18 Iteration.

An expression of the form loop *[testexp] { loopexp}* is an iteration expression. The first expression must be a number. The second expression is evaluated repeatedly, as long as *testexp* is not zero; the results of the *loopexp* are concatenated to form the result of the entire expression.

Examples:

i=0	loop[i#<2]{i "," i=i+1}	is "0,1,"
i=2	loop[i#<2]{i "," =i+1}	is ""i

5.3.2.5.3.19 Built-In Functions.

An expression of the form *func (arguments)* is a function call expression. The first expression must be one of the identifiers for the predefined MDL functions. The *arguments* part is a list of zero or more expressions, separated by commas. The result is a string value computed from the argument values by the predefined function.

Each predefined MDL function expects a certain number of arguments. It is an error if the *arguments* list does not contain the expected number of arguments.

MDL functions do not change their arguments. They always construct a new value completely separate from their arguments.

There is no way to add function definitions to a program. The predefined functions are the only functions available. The following paragraphs present a description of these functions.

5.3.2.5.3.19.1 The Alphatrim Function.

This function requires one argument, which must be a string. It returns a copy of the string with all leading and trailing white space removed.

Examples: (SPACE characters are represented by '.')

alphatrim("abcd")	is "abcd"
alphatrim("abcd...")	is "abcd"
alphatrim("...abcd")	is "abcd"
alphatrim("..abcd..")	is "abcd"

5.3.2.5.3.19.2 The Fail Function.

This function takes one argument, which must be a string. It causes MDLMAP to print the string as an error message, and then abort execution.

5.3.2.5.3.19.3 The Getenv Function.

This function takes one argument, which must be a string. It returns the value of the environment variable named by the string. If the current environment does not contain the named variable, the *empty* value is returned.

Examples:

getenv("SHELL")	is (probably) "/bin/csh"
getenv("UNDEF")	is (probably) *empty*

5.3.2.5.3.19.4 The Isnum Function.

This function takes one argument, which must be a string. It returns 1 if the argument is a number 0, otherwise.

Examples:

isnum("123")	is 1
isnum("-123:)	is 1
isnum("123abc")	is 0
isnum("foo123")	is 0

5.3.2.5.3.19.5 The Justleft Function.

This function requires two arguments. The first argument must be a string; the second argument must be a number. It returns a copy of the string argument, appending SPACE characters as necessary to make the returned value at least as long as specified by the second argument.

Examples: (SPACE characters are represented by '.')

justleft("abcd", 6)	is "abcd.."
justleft("abcd", 2)	is "abcd"
justleft("", 2)	is ".."

5.3.2.5.3.19.6 The Justright Function.

This function requires two arguments. The first argument must be a string; the second argument must be a number. It returns a copy of the string argument, pre-pending SPACE characters as necessary to make the returned value at least as long as specified by the second argument.

Examples: (SPACE characters are represented by a '.')

justright("abcd", 6)	is "..abcd"
justright("abcd", 2)	is "abcd"
justright("", 2)	is "..."

5.3.2.5.3.19.7 The Numtrim Function.

This function requires one argument. If this argument is a number, then it returns a copy of the number with all leading zeros and plus sign characters removed. If the argument is not a number, then numtrim simply returns a copy of the string.

Examples:

numtrim("00123")	is "123"
numtrim("+00123")	is "123"
numtrim("-00123")	is "-123"
numtrim("999")	is "999"
numtrim("abcd")	is "abcd"

5.3.2.5.3.19.8 The Tagct Function.

This function requires an empty argument list. It returns the number of records read from the input text, including those rejected because they did not match any record type or had too many fields.

5.3.2.5.3.19.9 The Truncate Function.

This function requires two arguments. The first argument must be a string; the second argument must be a number. It returns a copy of the string argument, removing characters from the right as necessary to make the returned string no greater than that specified by the second argument.

It is an error if the second argument is less than zero.

Examples:

truncate("abcde", 2)	is "ab"

truncate("abcde", 8) is "abcde"

5.3.2.5.3.19.10 The Undefined Function.

This function requires an empty argument list. It returns the *undefined* value.

5.3.2.5.3.19.11 The Warn Function.

This function requires one argument, which must be a string. It causes MDLMAP to write the string as a warning message. It returns the *empty* value.

5.3.3 Command-Level Interface to MDLMAP

MDLMAP runs as an independent program. When executed, it expects to receive command-line arguments telling it where to find its map file and input text, and where to write the output it produces. When MDLMAP terminates, it returns an exit status code to indicate whether errors were encountered during execution. Together the arguments and exit status make up the command-level interface to MDLMAP, which is described in this section.

This section uses the following typographical conventions:

a. Typewriter type represents output from the computer.
b. **Bold** type represents something that must be typed exactly.
c. *Italic* type represents a placeholder for something that must be provided.
d. Items in square brackets (e.g., [item]) are optional; they may be included or omitted.

5.3.3.1 Arguments to MDLMAP.

The usual way to invoke MDLMAP is through the following command:

mdlmap *mapfile*

Mapfile is the name of the file containing the MDL program to be executed. By default, MDLMAP expects to read its input text from the standard input, and write its output to the standard output. This behavior may be altered through the command-line options described below.

The complex syntax for invoking the MDLMAP program is:

mdlmap [-dtv][-m mapfile][**-i** input][**-o** output][mapfile [input [output]]]

Arguments that do not begin with a '-' character are interpreted by position. The first such argument is the name of the map file, the second is the name of the input text file, and the third is the name of the output file. For example:

mdlmap *mfile* map file in *mfile*, input text from the standard input, output to the standard output.

mdlmap *mfile ifile* map file in *mfile*, input text from *ifile*, output to the standard output.

mdlmap *mfile ifile ofile* map file in *mfile*, input text from *ifile*, output to *ofile*.

Arguments that begin with a '-' character may appear in any order as long as they all precede the positional arguments described above. These arguments have the following meanings:

-d Display (voluminous) information about what MDLMAP is doing and why it is doing it.

-i *inputfile* Obtain the input text from *inputfile*.

-m *mapfile* Obtain the map file from *mapfile*. If this argument is '-', obtain the map file from the standard input.

-o *outputfile* Write the result of the output expression to *outputfile*.

-t Check the map file for MDL errors. Do not process any output text.

-v Print the identification of this version of MDLMAP.

Certain arguments may be supplied by position or by option-letter. This means that there are several ways to specify the same behavior. For example, all three of the following commands will obtain the map file from **m1**, read the input text for **i2**, and write the result to **o3**.

```
mdlmap m1 i2 o3
mdlmap -o o3 -i i2 -m m1
mdlmap m1 < i2 > o3
```

It is possible to obtain both the map file and the input text from the standard input. For example, the following will work properly, provided that the map file does not contain any extraneous text following the MDL output expression:

```
cat m1 i2 | mdlmap -> o3
```

5.3.3.2 Exit Status Returned by MDLMAP.

When MDLMAP terminates, it returns an exit status code to the program that invoked it. This code allows the invoking program to determine whether MDLMAP encountered any errors during it execution, and if so, the nature of the errors.

MDLMAP returns one of the following six exit codes:

a. No errors were encountered during processing.
b. Errors occurred while compiling the MDL map file.
c. Errors occurred while parsing the input text file.
d. Errors occurred while evaluating the output expression.
e. The command-line arguments to MDLMAP were not valid.
f. An unknown, internal error occurred.

5.3.4 An MDL Tutorial

This section is a brief introduction to writing MDL programs. We describe the typical use of MDLMAP, introduce a specific, hypothetical task, and provide MDL map files for several variations on this task.

5.3.4.1 Automatic Message Generation.

MDLMAP is generally used as a part of a process known as *automatic message generation*. In this process, information is extracted from one or more operational databases, then reformatted to form part or all of a USMTF or ADatP-3 message.

MDLMAP is only a part of the automatic message generation process. The person in charge of the operational database is responsible for creating the *report generator*, which performs the data extraction. This report generator must produce its output in the "records-of-fields" format, which MDLMAP accepts as its input. Once the report generator is ready, the person in charge of MDLMAP must write the MDLMAP map file to transform the report generator's output into the desired message text format. The two programs, executed in sequence, produce message text containing information from the current state of the database.

5.3.4.2 A Typical Task: Generating TACELINT Messages.

For the remainder of this section, we will assume that our task is to generate TACELINT messages from the MTF standard. These messages are used to report time-critical electronic intelligence (ELINT) information. Figure 5-7 contains a brief description of the segments, sets, and fields in this message type. (For a complete description, refer to Joint Publication 6-04, the USMTF standard document.)

Because this is merely an example, and not a true TACELINT message generator, we will omit many of the details that would otherwise be required. We will assume that there is a database containing the ELINT information we want to report; however, we will not be concerned with its internal structure. We will also assume that the report generators exist but will not be concerned with their implementation. In our examples, we will simply state, "the output of the report

generator is..." without supplying complete details of how that output might be generated. Finally, we will ignore many of the fields that should be part of a TACELINT message, including only those necessary for the tutorial examples.

5.3.4.2.1 Example #1: Generating a Single Contact**.**

We begin with a simple task. Suppose we want to generate a Signal Operating Instructions (SOI) and Emitter Location (EMLOC) set for a single contact. We assume that our output generator produces a single untagged record for the contact containing all of the fields we require. The MDL input specification for this record is listed as follows:

```
    separator=',';
    terminator='\n';
    rec contact =
        tsi         //target signal identifier (SOI.1)
        dtime       //detection tiem (SOI.2)
        1time       //time lost (SOI.3)
        enot        //ELINT notation or sorting code (SOI.4)
        emdesig     //emitter designation (SOI.5)
        ldcat       //emitter location data category (EMLOC.2)
        loc;        //location (EMLOC.3)
input = contact ;
```

Generating this kind of input record might require nothing more than a qualified retrieval from the database table containing contact information. The SQL statement executed by the report generator is:

SELECT TSI, DTIME, LTIME, ENOT, EMDESIG, LDCAT, LOC
FROM table WHERE condition

TACELINT Format Summary

The TACELINT message is used to report time-critical operational electronic intelligence information. The sequence of sets in the message format is described in the following table.

occurrence	set ID	field occurrence	set format name
C	EXER	/M/O//	exercise information
C	OPER	/M/O/O/O//	operation identification data
M	MSGID	/M/M/O/O/O/0//	message identification
*O	REF	/M/M/M/M/O/O/*O//	reference
C	AMPN	/M//	amplification
C	NARR	/M//	narrative information
C	COLLINFO	/C/C/C/C//	collector information
[M	SOI	/M/M/M/M/M/C/C/C/C/C//	ELINT operational information
[*C	EMLOC	/M/M/M/C/C/C/C//	emitter location
[*C	PRM	/M/M/O/M/C/M/C/C/O//	signal analysis information
[*O	PLATID	/M/M/M/M/M/O/O/O//	platform identity
C	DECL	/M//	declassification data

The occurrence code "C" represents "conditional," "M" is "mandatory," "O" is "optional," and "*" is "repeatable." The "[" code indicates that the sets SOI, EMLOC, PRM, and PLATID may be repeated as a segment for reporting multiple signals.

Descriptions of some fields in the message are given below:

set and field #	field description
SOI.1	target signal identifier--code which identifies the detected signal
SOI.2	detection time--date-time group for time signal first detected
SOI.3	time lost--date-time group for time signal last heard
SOI.4	ELINT notation
SOI.5	emitter designation--nickname assigned to emitter
EMLOC.1	data entry--number correlating EMLOC and PRM sets
EMLOC.2	location data category--code describing emitter location (precise, estimated, etc.)
EMLOC.3	location
PRM.1	data entry--number correlating EMLOC and PRM sets
PRM.2	radio frequency
PRM.3	RF operational mode
PLATID.1	ship control number
PLATID.2	platform type--ship, aircraft, submarine, etc.
PLATID.3	ship type, or submarine type, or aircraft model

```
For a complete description of the TACELINT message, consult JCS
Publication 6-04.
```

Figure 5-8. An Abridged Description of the TACELINT Message

where *table* denotes the database table containing contact information, and *condition* is a test that eliminates all but a single row in that table. Here is a sample record produced by the report generator meeting the MDL input specification:

21345, 060821Z, 060821Z, XXXXX, LOUDMOUTH,E,LC:435240.5N0751826.4W

To form the output, we simply rearrange the input fields adding the set identifiers and field separators as necessary. The MDL output expression that transforms the input record into the SOI and ELOC sets we desire is shown as follows:

```
output = contact {
    'SOI/' tsi '/' dtime '/' ltime '/' enot '/' emdesig '//\n'
    'EMLOC/-/' ldcat '/' loc '//\n'};
```

Given the input record and MDL program above, MDLMAP produces the following two sets as its output:

SOI/21345/060821Z/060821Z/XXXXX/LOUDMOUTH//

EMLOC/-/E/LC:435240.5N0751826.4W//

5.3.4.2.2 Example #2: Generating Multiple Contacts**.**

Our ELINT database is sure to contain many contacts, so we will probably have to report several contacts in a single message. Using the approach in the previous example, we would need to run MDLMAP and the report generator once per contact. A better approach is to run the report generator once, extracting information for all of the contacts to be reported, and then run MDLMAP once to generate an SOI and EMLOC set for each contact. This example demonstrates this approach.

We assume that we want to generate a message reporting all contacts detected by a particular ELINT collector on a particular day. The report generator executes this query:

SELECT COLL,TSI,DTIME,LTIME,ENOT,EMDESIG,LDCAT,LOC

FROM *table* WHERE COLL=*code* AND *time-condition*

The *code* denotes the ELINT collector of interest, and *time-condition* is a test that excludes all rows not corresponding to the date of interest. Here is a sample of the output from the report generator:

AA,21345,060821Z,060821Z,XXXXX,LOUDMOUTH,E,LC:435240.5N0751826.4W

AA,21346,061021Z,-,XXXXX,FROGGER,E,LC:435240.5N0751826.4W

This input specification in our MDL program must change to accommodate the new field and the possibility of multiple input records. The new grammar is demonstrated as follows:

```
separator=',';
terminator='\n';
rec contact =
     coll          // denotes the ELINT collector (COLLINFO.1)
     tsi           // target signal identifier (SOI.1)
     dtime         // detection time (SOI.2)
     1time         // time lost (SOI.3)
     enot          // ELINT notation or sorting code (SOI.4)
     emdesig       // emitter designation (SOI.5)
     1dcat         // emitter location data category (EMLOC.2)
     loc;          // location (EMLOC.3)
input = rep { contact };
```

We must also change the output expression so that it executes the contact qualification expression once per input record. The new output expression is:

```
output = rep { contact {
'SOI/' tsi '/' dtime '/' 1time '/' enot '/' emdesig '//\n'
'EMLOC/-/' 1dcat '/' loc '//\n'}};
```

When we run MDLMAP on the sample input records above, it produces the following sets as its output:

```
SOI/21345/060821Z/060821Z/XXXXX/LOUDMOUTH//
EMLOC/-/E/LC:435240.5N0751826.4W//
SOI/21346/061021Z/-/XXXXX/FROGGER//
EMLOC/-/E/LC:435240.5N0751826.4W//
```

5.3.4.2.3 Example #3: Generating a COLLINFO Set**.**

We can extend the previous example by generating a COLLINFO set to describe the source of the reported contacts. A message contains a single COLLINFO set no matter how many contacts are

reported. We modify the output expression from the previous example to generate this initial set, as follows:

```
output =
// Generate the initial COLLINFO set, using info from the 1st
record
COLLINFO '/' rep[1].contact.coll '//\n'
// Now generate one segment per input record
rep { contact {
'SOI/' tsi '/' dtime '/' 1time '/' enot '/' emdesig '//\n'
'EMLOC/-/' 1dcat '/' loc '//\n'}};
```

Using the new output expression and the input records from the previous example, MDLMAP produces the following output text:

```
COLLINFO/AA//
SOI/21345/060821Z/060821Z/XXXXX/LOUDMOUTH//
EMLOC/-/E/LC:435240.5N0751826.4W//
SOI/21346/061021Z/-/XXXXX/FROGGER//
EMLOC/-/E/LC:435240.5N0751826.4W//
```

5.3.4.2.4 Example #4: Generating Repeated EMLOC Sets**.**

In the previous examples, we have assumed that we wish to produce exactly one EMLOC set for each source. However, the EMLOC set may be repeated in order to report multiple locations for a source, or it may be omitted entirely. In this example, we show how to generate repeated EMLOC sets.

We assume that the *target signal identifier (tsi)* and *detection time (dtime)* fields uniquely identify a contact; that is, if two records have the same *tst* and *dtime* fields, then they must contain information about the same contact. In order to group related records together, we add an ORDER clause to the report generator's SQL query, which now reads as follows:

```
SELECT COLL,TSI,DTIME,LTIME,ENOT,EMDESIG,LDCAT,LOC
FROM table WHERE COLL=code AND time-condition
ORDER BY TSI,DTIME
```

The output of the report generator does not change, except that there may be several records for the same contact. For example, the report generator may produce two records for contact #21345, one for contact #21346, and one for contact #21347, as follows:

```
AA,21345,060821Z,060824Z,XXXXX,LOUDMOUTH,E,LC:435240.5N0751826.4W
AA,21345,060821Z,060824Z,XXXXX,LOUDMOUTH,E,LC:435240.9N0751826.4W
AA,21346,061021Z,061044Z,XXXXX,FROGGER,F,LC:435233.2N0751823.3W
AA,21347,061044Z,061045Z,XXXXX,PACMAN
```

We assume that when there is no location information about a contact, the report generator omits those fields. In the input records above, it is up to us to notice that the first two records refer to a single contact, and then to produce one SOI and two EMLOC sets in that segment. We also must

notice that the last record contains no location fields, and then produce an SOI set with no EMLOC set in that segment. Our output should be:

```
COLLINFO/AA//
SOI/21345/060821Z/060824Z/XXXXX/LOUDMOUTH//
EMLOC/-/E/LC:435240.5N0751826.4W//
EMLOC/-/E/LC:435240.9N0751826.4W//
SOI/21346/061021Z/061044Z/XXXXX/FROGGER//
EMLOC/-/F/LC:435233.2N0751823.3W//
SOI/21347/061044Z/061045Z/XXXXX/PACMAN//
```

The input specification in this example is the same as in the previous example. Only the output expression needs to be changed. We add two conditional expressions. The first conditional expression suppresses the SOI set if the target signal identifier and detection time match those of the previous record. The second conditional expression suppresses the EMLOC set if the current record does not contain fields for this set. The new output expression is as follows:

```
output =
//Generate the initial COLLINFO set, using info from the 1st
record
COLLINFO '/' rep[1].contact.coll '//\n'
// The 1st record can't continue the previous contact
lastTSI=''
lastDTIME=''
//Generate one segment per input record
rep { contact {
// If tsi and dtime match fields in previous, skip SOI set
(tsi != lastTSI || dtime != lastDTIME) ?
'SOI/' tsi '/' dtime '/' 1time '/' enot '/' emdesig '//\n' :
       ''
// If no location info in this record, skip EMLOC set
(1dcat != '') ?
'EMLOC/-/' 1dcat '/' loc '//\n' :
       ''
// Remember tsi and dtime fields to compare against next record
lastTSI=tsi
lastDTIME=dtime
  }};
```

5.3.4.2.5 Example #5: Generating a Complete TACELINT Message**.**

In the previous examples, we have assumed that the report generator executes a single SQL query and produces a single type of record containing all the information we need about a particular contact. MDLMAP is capable of more powerful processing if the report generator executes multiple queries, producing different kinds of records. In this example, we will show how to construct a complete TACELINT message from information extracted by four SQL queries.

These are the tasks to be performed:

1. Generate a MSGID set, a DECL set, and (optionally) a COLLINFO set from the information contained in an initial *header* record.
2. Generate one SOI segment for each contact reported in a series of *soi* records.
3. Generate the EMLOC sets for each segment from a series of *emloc* records.
4. Generate PRM sets for each segment from a series of *prm* records. We must create *data entry* numbers for the EMLOC and PRM sets to show correspondences between the two.
5. Generate PLATID sets for each segment from a series of *platid* records.

To create the five kinds of input records for MDLMAP, we could run a single report generator that executed five SQL queries, one per record. Or, we could run five separate report generators, each executing a single query, and concatenate the results. Or, because the *header* record contains information that might not be part of our database, we might generate the *header* record with one program and the other records with the report generator. In all of these cases, MDLMAP receives the same input, and so produces the same result.

The MDL input specification for this example is presented below. It defines the five input record types and specifies the sequence in which these records must appear.

```
separator=',';
terminator='\n';
rec header 'H' =
     orig        // message originator (MSGID.2)
     serialnum   // message serial number (MSGID.3)
     month       // month name (MSGID.4)
     qual        // message qualifier [amplification, etc]
                 (MSGID.5)
     qualsnum    // original message serial number (MSGID.6)
     coll        // collector identifier (COLLINFO.1)
     decl;       // declassification date (DECL.1)
rec soi 'S' =
     tsi         // target signal identifier (SOI.1)
     dtime       // detection time (SOI.2)
     ltime       // time lost (SOI.3)
     enot        // ELINT notation or sorting code (SOI.4)
     emdesig;    // emitter designation (SOI.5)
rec emloc 'E'
     tsi         // target ID [matched to soi.tsi]
     dtime       // detection time [matched to soi.dtime]
     ldcat       // emitter location category (EMLOC.2)
     loc;        // location (EMLOC.3)
rec prm 'P' =
     tsi         // target ID [matched to emloc.tsi]
     dtime       // detection time [matched to emloc.dtime]
```

```
     loc         // location [matched to emloc.loc]
     freq        // radio frequency (PRM.2)
     mode        // RF operational mode (PRM.3)
     pfreq       // pulse interval/frequency (PRM.4)
     pdur;       // pulse duration (PRM.6)
rec platid 'PL' =
     tsi         // target ID [matched to soi.tsi]
     dtime       // detection time [matched to soi.dtime]
     shipnum     // ship control number (PLATID.1)
     ptype       // platform type detected (PLATID.2)
     stype       // ship type (PLATID.3)
     sclass      // ship class name (PLATID.4)
     sname;      // ship name (PLATID.5)
input =
     header                  // a single header record
     sois { soi }            // followed by any number of soi
                             recs
     emlocs { emloc }          // followed by any number of emloc
                             recs
     prms { prm }            // followed by any number of prm
                             recs
     platids { platid };     // followed by any number of platid
                             recs
```

The following is an example of the input records matching this input specification:

```
H,VMAQ1,0614024,JUN,-,-,AA,31DEC92
S,21345,060821Z,060824Z,XXXXX,LOUDMOUTH
S,21346,061021Z,061044Z,XXXXX,FROGGER
S,21347,061044Z,061045Z,XXXXX,PACMAN
E,21345,060821Z,E,LC:435240.5N0751826.4W
E,21345,060821Z,E,LC:435240.9N0751826.4W
E,21346,061021Z,F,LC:435233.2N0751823.3W
P,21345,060821Z,LC:435240.5N0751826.4W,00985.5MHZ,D,PRI:001085.98
7,-
,PD:0.450
P,21345,060821Z,LC:435240.5N0751826.4W,-,-,PRI:000521.162,-,-
P,21345,060821Z,LC:435240.5N0751826.4W,-,-,PRI:000564.735,-,-
P,21345,060821Z,LC:435240.9N0751826.4W,00985.5MHZ,D,PRI:001055.98
7,-
,PD:0.400
PL,21345,060821Z,22022,SHIP,DD,SPRUANCE,CUSHING
PL,21346,061021Z,-,ACFT,F15,-,-
```

The output expression presented below produces the TACELINT message in several stages. First, it creates the MSGID and COLLINFO sets using the technique described in section 5.3.4.2.3. Then, it loops through the *soi* records, producing one SOI segment for each record. For each SOI segment, it first prints the SOI set. Then, it loops through the *emloc* records, producing an EMLOC set for each record that matches the current segment. It does the same for

the *prm* and *platid* records. Finally, when all SOI sets have been produced, it writes the DECL set.

```
output =
// Produce MSGID set
'MSGID/TACELINT/' header.orig '/' header.serialnum '/'
header.month '/' header.qual '/' header.qualsnum '//\n'

// Produce COLLINFO set if collector ID supplied
(header.coll != '') ? ('COLLINFO/' header.coll '//\n') : ('')
// Produce one SOI segment for each soi record
sois {
// Produce the SOI set
'SOI/' soi.tsi '/' soi.dtime '/' soi.ltime '/'
soi.enot '/' soi.emdesig '//\n'

// Produce the EMLOC sets, generating data entry numbers
de = 1
emlocs { emloc {
(tsi==soi.tsi && dtime==soi.dtime) ?
('EMLOC/' de '/' ldcat '/' loc '//\n' de=de+1)
          ('')
     }}
// Produce the PRM sets, generating data entry numbers
prms { prm {
(tsi==soi.tsi && dtime==soi.dtime) ?
(de = 1 pde = '-'
emlocs {
(emloc.tsi==soi.tsi && emloc.dtime==soi.dtime) ?
(de=de+1
(emloc.loc==loc) ? pde=de : '')
               ('')}
'PRM/' pde '/' freq '/' mode '/' pfreq '/' pdur '//\n')
       ('')
     }}
// Produce the PLATID sets
platids { platid {
(tsi==soi.tsi && dtime==soi.dtime) ?
('PLATID/' shipnum '/' ptype '/' stype '/'
sclass '/' sname '//\n')
          ('')
     }}
   }
// Produce the DECL set
'DECL/' header.decl '//\n'
   ;
```

When executed on the sample input records, MDLMAP produces the following output:

```
MSGID/TACELINT/VMAQ1/0614024/JUN/-/-//
COLLINFO/AA//
```

```
SOI/21345/060821Z/060824Z/XXXXX/LOUDMOUTH//
EMLOC/01/E/LC:435240.5N0751826.4W//
EMLOC/02/E/LC:435240.9N0751826.4W//
PRM/01/00985.5MHZ/D/PRI:001085.987/-/PD:0.450//
PRM/01/-/-/PRI:000521.162/-/-//
PRM/01/-/-/PRI:000564.735/-/-//
PRM/02/00985.5MHZ/D/PRI:001055.987/-/PD:0.400//
PLATID/22022/SHIP/DD/SPRUANCE/CUSHING//
SOI/21346/061021Z/061044Z/XXXXX/FROGGER//
EMLOC/01/F/LC:435233.2N0751823.3W//
PLATID/-/ACFT/F15/-/-//
SOI/21347/061044Z/061045Z/XXXXX/PACMAN//
DECL/31DEC92//
```

5.4 BIT-ORIENTED MESSAGE (BOM) TO CHARACTER-ORIENTED MESSAGE (COM) TRANSLATOR

5.4.1 Identification

The paragraph describes version 1.0, release 1 of the BOM-to-MTF Translator software CSCI for the Army COE Message Block.

5.4.2 System Overview

The purpose of the Army COE message block software is to handle the dissemination and generation of various message sets such as the USMTF, ACCS, and BOM messages. The primary components of the message block are: the Message Parser Module; the Message Generation Module; the Normalization Library; the DCE-based Journaling Server; and the BOM-to-MTF message translator. The purpose of the message parser system is to process inbound messages, extracting information that is pertinent to the user. The purpose of the message generation software is to provide a GUI for the generation of messages. The purpose of the Normalization software is to convert the message data into a format usable by the host application software and vice versa. The purpose of the DCE Journaling Server is to provide an operational message journal for both inbound and outbound messages. The purpose of the BOM-to-COM software is to convert Bit-Oriented Messages to Character-Oriented Messages and Character-Oriented Messages to Bit-Oriented Messages.

5.4.3 Document Overview

The purpose of this paragraph is to describe the BOM/COM translator software CSCI, its functionality, the API, and its rules for usage. This paragraph contains the following sections: Reference Documents, Execution Procedures, and Error Messages.

5.4.4 Reference Documents

Document Number	**Title**
WP94B0000076	The Joint Message Analysis and Processing System (JMAPS) User's Manual

5.4.5 Installation

5.4.5.1 Installation of the Message Format Data Definition Database. The Message Format Data Definition (MFDD) database must be installed prior to executing the BOM/COM translator. Install your system-specified database, i.e., INFORMIX, Sybase, etc. Then, access the MFDD directory from the BOM/COM installation tape. This directory contains the Structured Query Language (SQL) scripts necessary to create the MFDD database and MFDD tables. Next, load the database. Once the MFDD database is installed, proceed with the installation of the BOM/USMTF translator.

5.4.5.2 Installation of the BOM/COM Translator. To install the BOM/COM translator, un-tar the software from the installation tape. Under the MTS directory, the following two sub-directories will exist: B2C and C2B. The B2C directory contains the BOM/COM CSCI. The C2B directory contains the COM/BOM software CSCI. The CSCIs will be configured for the requested platform (INFORMIX, Sybase, etc.). To reconfigure for a different platform, perform the following tasks: type **make -f<makefile.informix** or **makefile.sybase>**, depending on which database you are using.

5.4.6 Execution Procedures

5.4.6.1 Initialization.

The only initialization required for the BOM/COM translator is for the database software to be loaded and executing. If your installed database is not named 'mfdd', then set an environment variable DB to the name of the installed database.

5.4.6.2 User Inputs.

The calling sequence for the BOM/COM and COM/BOM translators is shown as follows:

B2C<client/server><filename> , **where client specifies** a DD173 header and server specifies a Comm Free Header where filename is the file containing the Bit Oriented Message.

C2B<filename> **, where filename is the file containing** the Character Oriented Message.

5.4.6.3 System Inputs.

The system inputs to the BOM/COM translator consist of the MFDD database which was installed under the user-specified DBMS (see Section 5.4.5).

5.4.6.4 Termination.

The BOM/COM translator software is a callable function. Use of the BOM/COM translator places no special termination constraints on the calling application.

5.4.6.5 Outputs.

The output of the B2C function is a file containing the converted COM. The output of the C2B function is a file containing the converted BOM.

5.4.7 ERROR MESSAGES

The following is a listing of the error messages output by the B2C and C2B software, the associated meaning of the message, and the action taken when each message appears.

<TBD>

5.4.8 "Customization of the BOM/COM Translator"

<TBD>

5.4.9 "Example Usage of the BOM/COM Translator"

<TBD>

5.5 MESSAGE JOURNALING SERVER

5.5.1 Identification

This section describes the journaling module of the CMP Version 1.2.x.x for the COE Message Block. The purpose of the DCE Journaling server is to provide an operational message journal for both inbound and outbound messages, and to allow lookup and retrieval of information contained in the message, or retrieval and manipulation of the message.

5.5.2 Section Overview

This section contains the following subsections: Reference Documents, Installation Procedures, Execution Procedures, and Error Messages.

5.5.3 Reference Documents

Document Number	**Title**
TR-32-95	The BOM-to-COM Translator User's Manual

5.5.4 Installation Procedure - Refer to Appendix A of this document.

5.6 MTF STAND-ALONE TOOLS

Some of the services that the Message Parser performs are available as separate tools. These tools are independent programs, each performing a single function. They are invoked by a user from a shell prompt in the same way that standard UNIX utilities (for example, *ls* and *cat*) are invoked. They do not depend on or interact with the parser Message Processor (MP) or User Interface (UI) described in the previous sections. These tools provide the following services:

a. Validate a message and output an error report
b. Execute a JQL query on a message and output the query results
c. Output a message in report format
d. Generate a message from data extracted from a database.

5.6.1 The MTFVAL Tool

The **mtfval** tool is a message-validation tool. It accepts a single message on standard input or a list of message file names as its command-line arguments. This tool validates each message and produces an error report for each message that contains one or more errors. No output is produced for a correct message.

The **mtfval** tool accepts the following optional command-line arguments:

> **-v** If **mtfval** is reading from files named on the command line, then the name of each input file is printed before the error report for that file. File names are printed for both correct and incorrect messages.
>
> **-o** outputfile

All output is written to *outputfile* instead of the standard output.

5.6.2 The MTFXTRACT Tool

The **mtfxtract** tool executes JQL queries on messages. The queries and messages are specified in the command-line arguments, **mtfxtract** processes its arguments from left to right. Each argument is one of the following:

> **-q** means that the next argument contains the text of a query. This becomes the current query. (It is necessary to quote the query text to keep the shell from expanding meta characters and breaking them into separate words.)
>
> **-qf** means that the next argument is the name of a file that contains a query. This becomes the current query.
>
> **-o** means that the next argument is the name of an output file. **mtfxtract** will write all subsequent output to this file. The current contents of the file, if any, are lost.

Anything else is taken as the name of a file that contains a message. **mtfxtract** applies the current query to the message and outputs the query results. It is an error to supply a message if there is no current query.

The single character "-", when used in the place of a query or message file name, tells **mtfxtract** to read the query or message from its standard input instead of from a file.

5.6.3 The MTFREPORT Tool

The **mtfreport** tool prints a message in *report format*. It expects a single command-line argument, which is the name of a file containing a message. It writes the message in report format to its standard output. If the single character "-" is given in the place of a filename, **mtfreport** reads the message from its standard input instead of from a file.

6 NOTES

6.1 ACRONYM LIST

Acronym	Definition
ACC	Air Combat Command
ACCS	Army Command and Control System
ACOE	Army Common Operating Environment
ACP	Allied Communications Publication
ADatP-3	Allied Data Publication 3
AFATDS	Advanced Field Artillery Tactical Data System
AFB	Air Force Base
AI	Applications Interface
AMHS	Automated Message Handling System
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ATCCS	Army Tactical Command and Control Systems
ATOCONF	Air Tasking Order/Confirmation
ATTable	Application Trigger Table
AUTODIN	Automatic Digital Network
B2C	The BOM-to-COM translator
BFA	Battlefield Functional Area
BOM	Bit-Oriented Message
C2	Command and Control
C2B	The COM-to-BOM translator
C3I	Command, Control, Communications, and Intelligence
C4I	Command, Control, Communications, Computers, and Intelligence
C4IFTW	C4I for the Warrior
CASS	Common ATCCS Support Software
CCP	Communications Protocol Preprocessor
CDBS	Central Database System
CHS	Common Hardware and Software
CMP	Common Message Processor
COE	Common Operating Environment
COM	Character-Oriented Message
COMCAT	Character-Oriented Message Catalog
CPP	Communications Protocol Preprocessor
CSCI	Computer Software Configuration Item
CSS	CASS Segment Specification

CTAPS	Contingency Theater Automated Planning System
CUI	CMP User Interface
DBMS	Database Management System
DCE	Distributed Computing Environment
DII	Defense Information Infrastructure
DISA	Defense Information Systems Agency
DOD	Department of Defense
DOI	DSCS Operating Instructions
DSCS	Defense Satellite Communications Systems
DSSCS	Defense Special Security Communications Systems
DTG	Date Time Group
EAST_NORTH	Easting, Northing and Grid Zone
ELINT	Electronic Intelligence
EMLOC	Emitter Location
ERTM	ELINT Requirement Tasking Messages
GCCS	Global Command and Control System
GUI	Graphical User Interface
HP	Hewlett-Packard
HQ	Headquarters
HW	Hardware
IDL	Interface Design Language
IEW	Intelligence and Electronic Warfare
IMP	Inbound Message Processor
IPC	Inter-process Communication
JAMPS	Joint Automated Message Preparation System
JANAP	Joint Army, Navy, Air Force, Publication
JCS	Joint Chiefs of Staff
JMAPS	Joint Message Analysis Processing System
JMPS	Joint Message Preparation System
JQL	JMAPS Query Language
JUDI	Joint Universal Data Interpreter
LAT_LONG	Latitude/Longitude
Log	System Log
MCS	Maneuver Control System
MDL	Map Definition Language
MFDD	Message Format Definition Database
MGRS	Military Grid Reference System
MIL STD	Military Standard
MIO	Message Input/Output
MP	Message Processor
MsgBase	Message Base
MTS	Marine Tactical System

NATO	North Atlantic Treaty Organization
NCTSI	Navy Center for Tactical Systems Interoperability
NESEA	Naval Electronic Systems Engineering Activity
ODB	Operational Database
OMG	Outbound Message Generator
OSF	Open Software Foundation
OTH	Over the Horizon
PLA	Plain Language Address
PQL	Parser Query Language
PREP	Message Preparation
Pub. 6-04	Joint Publication 6-04
QBase	Query Base
QRBase	Query Report Base
RAM	Random Access Memory
RPC	Remote Procedure Call
SECS	Seconds
SOI	Signal Operating Instructions
SQL	Structured Query Language
SUN	SUN Microsystems
SW	Software
TAFIM	Technical Architecture for Information Management
TBM	Theater Battle Management
UI	User Interface
USAF	U.S. Air Force
USMTF	United States Message Text Format
USSID	United States Signals Intelligence Directive
WCCS	Wing Command and Control System

6.2 GLOSSARY

TERMS

Allied Data Publication 3: A NATO formatted text message standard.

automatic message generation: The process of populating the fields of a formatted message using information extracted from one or more operational databases.

comment: Descriptive text that is part of a program but has no effect on its behavior.

field: In formatted text, a sequence of characters delimited by a separator or terminator string.

grammar: In MDL, a definition of all of the valid sequences of records in the input text.

group: In MDL, an ordered sequence of elements to be found in the input text. The members of a group are either records or repetitions.

identifier: In MDL, a name supplied by the programmer.

Map Definition Language: A special-purpose language that defines a transformation between an input and an output text format.

map file: A program written in MDL.

number: In MDL, an integer represented as a sequence of digits, optionally preceded by a plus (+) or minus (-) sign.

operational database: A database containing information used by a tactical battle management system.

parse: The process of determining how a particular sequence of input records may be validly derived from an input grammar

parse tree: A data structure showing how a sequence of input records are combined into groups and repetitions according to the input grammar. It is constructed during the parsing process.

record: In formatted text, a sequence of fields ending in a terminator string.

repetition: In MDL, a repeating sequence of elements to be found in the input text. Each element of a repetition is always a group. Repetitions occur a the record level; tail repetitions occur within a particular record.

reserved word: In MDL, a name defined by the language which cannot be used as an identifier by the programmer.

scope: In MDL, the part of the program in which an identifier has a particular meaning assigned by the programmer.

separator: In formatted text, a sequence of characters signaling the end of one field and the beginning of another.

set: In the ADatP-3 and USMTF message standards, a particular type of record. The first field contains the name of the set type.

string: A sequence of characters enclosed in a pair of single quotes (') or double quotes (").

Structured Query Language: The *de facto* standard data definition and manipulation language used as a baseline by many database implementations.

tail group: In MDL, the elements of a tail repetition. The elements of a tail group are fields in an input record.

tail repetition: In MDL, a repeating sequence of elements to be found at the end of a single input record. See repetition.

terminator: In formatted text, a sequence of characters signaling the end of an input record.

United States Message Text Format: A DoD voice and formatted text message standard.

6.3 LIST OF DEFINITIONS

alternate initial sets: A choice of two or more sets, exactly one of which can be used as the first set in an instance of a given USMTF segment.

Backus-Naur Form: A notation system frequently used to represent the syntax of programming languages and other context-free grammars.

client: Any program which uses the parser to extract information from USMTF or OTH Gold messages.

data element: Any complete USMTF/OTH Gold or subordinate USMTF/OTH Gold component segment, set, or field that can be designated in JQL.

data element specifier: In JQL, the syntax used to reference a given USMTF/OTH Gold segment, set, or field.

data item: The information contents of a USMTF/OTH Gold data element.

date-time group: A standardized, formatted representation of the date and time at which a message was originally transmitted.

domain: see message domain.

field: In the MTF standard, a low-level data item containing formatted information within a set.

field format index reference number: An index into a Joint Pub. 60-40 table of field formats that identifies the allowable structure of that MTF field.

field use designator: In the MTF standard, one of a set of codes assigned to an ffirn that indicates the meaning of the data encoded in the field.

inline function: One of a limited number of function that may be applied to selected data elements in JQL for the purposes of retrieving information associated with or derived from the cited data rather than the cited data itself.

Joint Message Analysis Processing System (JMAPS): An integrated MTF processing system developed by MITRE; it will support the preparation, transmission, and receipt of the complete catalog of MTF messages, as well as automatic updates to changes in the standard, demand-driven parsing and validation to data feed arbitrary user

applications, and constraint-based reasoning capabilities.

JMAPS Query Language: A data manipulation language used to access MTF data and to direct partial message parsing and validation in JMAPS.

message base: A central repository of MTF messages definitions in an internal format that represents the totality of messages processed by the parser.

message domain: In JQL, the subset of the message base that represents the particular MTF from which data is retrieved.

message element: see **data element.**

message text format: In the MTF standard, the highest level data entity; it consists of a predetermined sequence of sets and segments determined uniquely by the identifying name of the message.

message type: Any instance of a MTF message having a given message name.

originator: The plain-language designator of the activity that was the point of origin of a transmitted message.

parse: In MTF processing, to locate and identify the significant structural components of a message.

placeholder: In a JQL shell, an ampersand character ('&') followed by an integer which shows where substitution of retrieved MTF data into a copy of a template will occur.

presentation number: In the sequence of possible sets making up a given MTF type, an integer value that is theoretically assigned to each set to illustrate its relative position in the message.

registration: The process of informing JMAPS of the information requirements of a new client application.

scope: In a JQL shell, the domain within which the substitution of a given collection of retrieved data elements into a text template occurs.

segment: In the MTF standard, a predetermined sequence of related sets and possibly other segments that are treated as a contiguous information group.

set: In the MTF standard, a predetermined sequence of related fields together with a name that indicates the relationship of the contained fields.

shell: The combination of one or more JQL statements together with one or more text templates into which MTF derived data will be substituted.

standard: In the context of this paper, Pub. 60-40, the document that defines the basic rules governing the structure and content of a MTF.

Structured Query Language: The *de facto* standard data definition and manipulation language used as a baseline by many commercial databases.

superuser: In UNIX, the user account with the ability to access and modify any file or resource. Also known as the **root** account.

template: In a JQL shell, text that contains placeholders for representing where retrieved MTF data may be substituted.

United States Message Text Format: A DOD voice and formatted text message standard.

validation: The verification that the data contained in the structural components of a given MTF obey the rules for content and use as stated in Pub. 60-40.

wildcard: The asterisk symbol (*), used in JQL's domain resolution expressions to denote the absence of a constraining value for some key feature of messages of interest.

APPENDIX A
INSTALLATION GUIDANCE
(NOTE: This is not necessary for DII COE use)

1. INTRODUCTION

The purpose of this appendix is to define hardware and software environmentals necessary to install individual, or all of the components of CMP.

2. POSSIBLE CONFIGURATIONS

2.1 Layout of distinct CMP Configurations

The following table is a layout of distinct CMP configurations and the physical segments needed to perform the functions.

	Config	Parser (JMAPS)	Msg Gen (JMPS)	Journaling	CMP User Int.	Discriminator	Norm	Other COE/Data Segs
1	**JMAPS**	x				x		#,T
2	**JMAPS/ JMPS/**	x	x			x		#,T
3	**JMAPS/ JMPS/ Journal**	x	x	x		x		#,*,$,T
4	**JMAPS/ JMPS/CUI**	x	x	x	x	x		#,*,$,T
5	**JMAPS/ JMPS/CUI/ NORM**	x	x	x	x	x	x	#,*,$,T
6	**JMPS**		x					#,T
7	**JMPS/ Journal**		x	x				#,*,$,T
8	**JMPS/CUI**		x	x	x	x		#,*,$,T
9	**JMPS/CUI/ Norm**		x	x	x	x	x	#,*,$,T
10	**JMPS/Norm**		x				x	#,T
11	**JMPS/Norm**		x	x			x	#,*,$,T

	/Journal							
12	**JMAPS/ Journal**	x		x		x		#,*,$,T
13	**JMAPS/ Norm**	x				x	x	#,T
14	**JMAPS/ Norm/ Journal**	x		x		x	x	#,*,$,T
15	**JMAPS/ Journal/CUI**	x		x	x	x		#,*,$,T

Legend:

x - Required
- COE Com Required
* - COE DCS Required
$ - COE DBMS Required
T - Message Tables Required
blank - Not required.

2.2 Installation Scenarios

The following is a description of the installation scenarios for each of the above configurations.

Index Resources

(1) Disk Space: 31MB
Shared Memory/Semaphores: See JMAPS Kernel Configurations

(2) Disk Space: 71MB
Shared Memory/Semaphores: See JMAPS Kernel Configurations

(3) Disk Space: 72MB
DCE: Name Space (/.:/jnl_int_`hostname`, /.:/jnl_int_group)
Shared Memory/Semaphores: See JMAPS Kernel Configurations

(4) Disk Space: 80MB
DCE: Name Space (/.:/jnl_int_`hostname`, /.:/jnl_int_group)
Shared Memory/Semaphores: See JMAPS Kernel Configurations

(5) Disk Space: 81MB
DCE: Name Space (/.:/jnl_int_`hostname`, /.:/jnl_int_group)
Shared Memory/Semaphores: See JMAPS Kernel Configurations

(6) Disk Space: 67MB

(7) Disk Space: 68MB
DCE: Name Space (/.:/jnl_int_`hostname`, /.:/jnl_int_group)

(8) Disk Space: 75MB
DCE: Name Space (/.:/jnl_int_`hostname`, /.:/jnl_int_group)

(9) Disk Space: 76 MB
DCE: Name Space (/.:/jnl_int_`hostname`, /.:/jnl_int_group)

(10) Disk Space: 68 MB

(11) Disk Space: 69 MB
DCE: Name Space (/.:/jnl_int_`hostname`, /.:/jnl_int_group)

(12) Disk Space: 32 MB
DCE: Name Space (/.:/jnl_int_`hostname`, /.:/jnl_int_group)

(13) Disk Space: 32MB

(14) Disk Space: 33MB
DCE: Name Space (/.:/jnl_int_`hostname`, /.:/jnl_int_group)

(15) Disk Space: 40MB
DCE: Name Space (/.:/jnl_int_`hostname`, /.:/jnl_int_group)

The following Kernel parameters are required for JMAPS and COE DBMS and COE DCS services to function together. These parameters apply to all configurations using the parser (JMAPS), or Journaling Server (which requires COE DCS and DBMS services).

JMAPS Kernel Configurations (/etc/system):

set shmsys:shminfo_shmmax=8388608

set semsys:seminfo_semmap=64

set semsys:seminfo_semmni=4096

set semsys:seminfo_semmns=200

set semsys:seminfo_semmnu=4096

set semsys:seminfo_semume=64

set shmsys:shminfo_shmmin=1

set shmsys:shminfo_shmmni=100

set shmsys:shminfo_shmseg=10

2.3 COE run-time dependencies

The following details COE run-time dependency segments that should be installed and running when before attempting to install the CMP

Index Segments

(1) Comm interface, Motif 1.2, X11R5

(2) Comm interface, Motif 1.2, X11R5

(3) Comm interface, Motif 1.2, X11R5, DCE

(4) Comm interface, Motif 1.2, X11R5, DCE

(5) Comm interface ,DCE, Motif 1.2, X11R5, DCE

(6) Comm interface, Motif 1.2, X11R5

(7) Comm interface ,Motif 1.2, X11R5, DCE

(8) Comm interface ,DCE, Motif 1.2, X11R5, DCE

(9) Comm interface ,DCE, Motif 1.2, X11R5, DCE

(10) Comm interface, Motif 1.2, X11R5

(11) Comm interface, DCE, Motif 1.2, X11R5

(12) Comm interface, DCE, Motif 1.2, X11R5, DBMS (COE DBMS)

(13) Comm interface, Motif 1.2, X11R5

(14) Comm interface, DCE, Motif 1.2, X11R5, DBMS (COE DBMS)

(15) Comm interface, DCE, Motif 1.2, X11R5, DBMS (COE DBMS)

3.0 Registration and Query

3.1 Registration of Client Applications

Prior to using the CMP, the information required by each client application must be described to the processor, by the user in a process called *registration*. Clients must specify the types of messages in which they are interested, the fields to be extracted from messages of these types, and the destination to which the extracted fields must be routed. This registration information may either be given to the processor at startup time in a client configuration file, or interactively through the user interface.

This section describes a hypothetical client application called ISUM, which collects various kinds of intelligence data and summarizes them in a report for its users. The ISUM program is not a part of the processor; in fact, it is not a real program at all. ISUM simply represents a typical client program that requires information from a message.

To do its job, ISUM needs some of the information present in TACELINT messages. These messages report time-critical operational electronic intelligence information. An abridged description of the message format is in Table A-1. Specifically, ISUM is interested in reports of surface ship emitters that have been precisely located. ISUM needs to now the signal ID, detection time, location, and ship responsible for these signals. The parser must extract the fields containing this information from incoming TACELINT messages and route the results to ISUM. Also, because ISUM is sensitive to errors in its input data, the parser must validate the incoming messages and inform ISUM if any errors are discovered. The sequence of sets in the message format is described in Table 3-2.

Table A-1. An Abridged Description of the TACELINT Message

Occurrence	Set ID	Field Occurrence	Set Format Name
C	EXER	/M/O//	exercise information
C	OPER	/M/O/O/O	operation identification data
M	MSGID	/M/M/O/O/O/O	message identification
*O	REF	/M/M/M/M/O/*O//	reference
C	AMPN	/M//	amplification
C	NARR	/M//	narrative information
C	COLLINFO	/C/C/C/C//	collector information
[M	SOI	/M/M/M/M/M/C/C/C/C/C//	ELINT operational information
[*C	EMLOC	/M/M/M/C/C/C/C//	emitter location
[*C	PRM	/M/M/O/M/C/M/C/C/O//	signal analysis information
[*O	PLATID	M/M/M/M/M/O/O/O//	platform identity

The occurrence category "C" represents "conditional"; "M" is "mandatory"; "O" is "optional"; and "*" is "repeatable". The "[" code indicates that the sets SOI, EMLOC, PRM, and PLATID may be repeated as a segment for reporting multiple signals. All of the sets in the segment describe the same emitter. Sets in different segments are logically independent.

Descriptions of some of the fields in the message are given in Table A-2.

Table A-2. Partial Field Description

Set and Field #	Field Description
SOI.1	target signal identifier - code that identifies the detected signal
SOI.2	detection time – date-time group for time signal first detected
EMLOC.2	location data category – code describing emitter location
EMLOC.3	location
PLATID.1	ship control number
PLATID.2	platform type – ship, aircraft, submarine, etc.
PLATID.3	ship type, or submarine type, or aircraft model
PLATID.4	ship class name, or submarine class name, or aircraft name
PLATID.5	ship name

In this section we will take the role of an ISUM system developer arranging to register ISUM with the parser. First, we must write a query that describes the fields we require and also specifies our validation requirements. Second, we must write a routing table entry to execute the query on new TACELINT messages and send the results to ISUM. In the following sections, we show how to write the query and routing entry, and then show how to give these entries to the parser using either the user interface or a client configuration file.

3.2 Writing the Query for ISUM**.**

Queries are written in Structured Query Language (SQL), which is a standard database query language. Queries describe (a) the conditions that the contents of a message must satisfy in order to be selected, and (b) the parts of a message which will be extracted and sent to an application.

A query is composed of several clauses. Here is a very brief description of the kinds of clauses used in the following example and the meanings.

a. The SELECT clause describes the message fields to be extracted and sent to the application. Specifying multiple fields asserts a logical relation between the selected fields, which will be reflected in the output. (This will be illustrated by the example appearing later in this section.)

b. The FROM clause describes the messages to be considered. Messages must be specified by message type, and may be further defined by originator or by date-time group.

c. The WHERE clause specifies a test that must be met by the message contents. Message data that fails this test is not included in the query results.

d. The FORMAT clause describes the output format of the selected message fields. Most queries use FORMAT TABLE, which organizes the output in the form of a relational table, suitable for importing into a database.

e. The VALIDATE clause describes the portions of the message contents to be validated. The

choices are to validate the entire message, or only the extracted fields, or nothing at all. In order to write the query for ISUM, we must consult the definition of the TACELINT message format. The first step is to specify the fields to be extracted from each message. ISUM needs the signal ID, detection time, location, and information about the ship emitting the signal. Studying the standard, we find that we need the following five fields:

(1)	SOI.1	target signal identifier
(2)	SOI.2	signal detection time
(3)	EMLOC.3	signal location
(4)	PLATID.3	ship type
(5)	PLATID.5	name of ship

All five fields will appear in the SELECT clause. These fields contain everything that ISUM needs. However, we may have to add fields after we write the WHERE clause; see below.

Because we are not writing an interactive query, we can omit the FROM clause. The Message Parser will apply this query to each TACELINT message individually, as it arrives. There is no need to tell the parser which messages to consider here.

Next, we must write the WHERE clause in order to restrict the signals that are reported to ISUM. We are only interested in precisely-located surface ship emitters. Consulting the standard again, we discover that the value "F" in the EMLOC.2 field (emitter location data category) tells us whether the signal has been precisely located. The value "SHIP" in the PLATID.2 field (platform type detected) tells us that the signal emitter is a surface ship. Putting these two together, we can write the following WHERE clause:

WHERE EMLOC.2="F" AND PLATID.2="SHIP"

It is a rule that every field mentioned in the WHERE clause must be included in the SELECT clause. We must add the above two fields to the five fields required by ISUM.

Next, we will write "FORMAT TABLE" to organize the extracted information into a relational table. ISUM will import this table into its internal database.

Finally, we write "VALIDATE ALL" to cause the parser to report any validation error in the incoming message. We want ISUM to reject information from an invalid TACELINT message, even if the specific fields extracted are valid.

We have to give our query a name. We will call it "isum-01." The complete text of the query appears as follows:

SELECT SOI.1,SOI.2,EMLOC.3,

PLATID.3,PLATID.5,

EMLOC.2,PLATID.2

WHERE EMLOC.2="F" AND PLATID.2="SHIP"

FORMAT TABLE or VALIDATE ALL;

When the *isum-01* query is executed on a TACELINT message, it will extract the information required by the ISUM program. For example, a sample TACELINT message is displayed in Figure A-1. This message contains four segments describing four separate emitters.

When we execute the *isum-01* query on this message, we get the five output rows displayed below:

"21400","010951Z","LS:435244N0751820W","DD","CUSHING","F","SHIP"

"21400","010951Z","LS:435244N0751820W","DD","BURKE","F","SHIP"

"21400","010951Z","LS:435244N0751822W","DD","CUSHING","F","SHIP"

"21400","010951Z","LS:435244N0751822W","DD","BURKE","F","SHIP"

"21401","010951Z","LS:435250N0751850W","DD","LEFTWICH","F",SHIP"

OPER/GRAVE/DIGGER//

MSGID/TACELINT/RHDIAAA/0401024/APR//

COLLINFO/BH//

SOI/21392/010945Z/0109487/EHIZZ/LOUDMOUTH/DQ//
EMLOC/01/P/LS: 43352400N0751826W/-/011.0T/22KM/8KM//
PRM/01/00895.5MHZ/D/PRI:001085.897/S/PD:0.540/STDY/-//
PRM/01/-/-/PRI:000521.162/S/PD:0.550//

segment #1

PLATID/11011/SHIP/DD/SPRUANCE/CUSHING/985/US

SOI/21395/010942Z/0100947Z/EHIZZ/HIGHBLOW/DQ//
EMLOC/01/F/LS:4335241N0751830W//
PLATID/-/AIR/F15/-/-//

segment #2

SOI/21400/010951Z/010953Z/EHIZZ/LOUDMOUTH/DQ//

EMLOC/01/F/LS:435244N0751820W//
EMLOC/02/F/LS:435244N0751822N//
PLATID/11011/SHIP/DD/SPRUANCE/CUSHING/985/US//

segment #3

PLATID/12200/SHIP/DD/BURKE/BURKE/022/US//

SOI/21401/010951Z/010951Z/EHIZZ/LOUDMOUTH/DQ// segment
EMLOC/01/F/LS:435250N0751850W// #4
PLATID/11012/SHIP/DD/SPRUANCE/LEFTWICH/984/US//

Figure A-1. A Sample TACELINT Message

Note that our query extracts nothing from the first two segments. The first segment is skipped because it reports an estimated position (EMLOC.2="P"). The second segment is skipped because it describes an aircraft emitter (PLATID.2="AIR").

Note that four output rows are produced for the signal reported in the third segment. The third segment gives two locations and two identifications for the signal. Recall that the SELECT clause in our query asserts that the selected values are related. The CMP produces all four possible combinations of values for this relation.

Note that the CMP does not produce all possible combinations of values in separate segments. The values extracted from separate segments describe separate emitters; they are not related, so the parser does not combine them. This is an example of the general rule that it does not produce combinations of values from unrelated segments (see Section 3.6.1.2.2).

3.2.1 Writing the Routing Table Entry for ISUM.

Creating the routing table entry is the second part of registering an application with the CMP. A routing table entry tells the processor that every time a particular type of message arrives, it should execute a query on the message and send the output to a client application.

To write the routing table entry for ISUM, we need to know five things:

a. The type of message to be processed. For ISUM, the message type is TACELINT.

b. Whether we want the incoming messages to be saved in the message repository. This is necessary if we later want to correct and resubmit invalid messages. For ISUM, we want to be able to do this.

c. The name of the query to execute on these messages. For ISUM, the query name is "isum-01."

d. The name of the client application program to receive the query output. When not using DCE this should usually be an absolute pathname. The name of the ISUM executable is "/usr/isum/isum."

e. The host name of the machine where the client program will be executed. ISUM executes on the same machine as the message processor, so we use "localhost" for this.

There are two different ways we can enter this information into the routing table; through the user interface, or through a client configuration file. We will see both methods in the following sections.

3.3 Registering ISUM with the User Interface**.**

As an ISUM developer, we will want to test the configuration to ensure that the correct information is extracted and routed to ISUM. By using the UI, we can quickly enter, test, and modify configuration information without having to restart the processor each time we want to make and test a change. In this section we show how an ISUM developer would create and test the ISUM registration. We will deliberately make a mistake, and show how to detect and correct it.

(The user interface is based on the Motif user environment. Readers not familiar with this environment might wish to turn to Section 3.7.2, which contains an overview of the environment and the terminology used to describe its use.)

We assume that the message processor is running and that we have invoked the user interface. The first step in registering ISUM is to create the new query. To do this using the parser UI, we follow these five steps:

Step 1 Using the menu bar, we choose Windows Queries, then Commands New. This displays a Queries(NEW) window.

Step 2 Enter the name of the query into the Query Name input field. We type **isum-01**.

Step 3 Enter the text of the query into the Query input field. We should type the text of the query appearing on page. Instead, we mistakenly leave out the PLATID.3 field in the SELECT clause.

Step 4 Press the Validate button to check for syntax errors in the query text.

Step 5 Press the OK button once all errors have been corrected.

The second step in registering ISUM is to create the routing table entry. To do this, we choose Windows Routing. This creates the Routing window.

There is no entry for TACELINT messages in the Routing window, so we must create one. We do this using the following four steps:

Step 1 Choose Commands New. This creates a Routing(NEW) window.

Step 2 Enter the name of the message type into the Message ID input field. We type "TACELINT".

Step 3 Turn on the Save Message check box. This tells the processor to save incoming TACELINT messages in the message journal.

Step 4 Press the OK button.

After we press the OK button, the Routing(NEW) window vanishes, and the new entry for TACELINT messages appears in the Routing window.

We have now added an entry for TACELINT messages into the routing table. Next, we must add ISUM to the list of clients requiring information from these messages. To do this, we follow these seven steps:

Step 1 Select the TACELINT list item in the Routing window (by clicking on it).

Step 2 Press the Applications button. This creates an Applications window, which displays the application list for TACELINT messages.

Step 3 Choose Commands New. This creates an Applications (NEW) window.

Step 4 Enter the name of the client application into the Application input field. This would ordinarily be the name of the ISUM executable file, "/usr/isum/isum." However, because we are testing, we enter "cat>isum.in" instead. This causes the query output to be saved in the file named "isum.in" in the current directory.

Step 5 Enter the name of the machine that runs the application into the Host input field. We type "localhost."

Step 6 Enter the query used to filter TACELINT messages routed to this application in the Query input field. We type "isum-01."

Step 7 Press the OK button.

When we press the OK button, the Applications(NEW) window vanishes, and the new application entry appears in the Applications window. To make this window vanish, press Close.

That is the final step in our registration of ISUM. Next, we want to test the registration by sending a TACELINT message and examining the output that would be routed to ISUM. Assume that the sample TACELINT message shown in Figure 3-4 is contained in a file named "tac1.msg" in the current directory. We can give this message to the processor with the command

mtf2parser tac1.msg. It will run the *isum-01* query on the message and send the result to the *cat* command, which creates the *isum.in* file. The contents of this file are shown in Figure A-2.

```
.MSGID TACELINT
.FROM RHDIAAA                                                    header
.TO DHDIAZZ/SYJ                                                  lines
   .DTG 011200z APR 93
   .ERRORS 0
   "21400","010951Z","LS:435244N0751820W","CUSHING","F","SHIP"
"21400","010951Z","LS:435244N0751820W","BURKE","F","SHIP"        extracted
"21400","010951Z","LS:435244N0751820W","CUSHING","F","SHIP"      data
   "21400","010951Z","LS:435244N0751820W","BURKE","F","SHIP"
   "21401","010951Z","LS:435250N0751850W","LEFTWICH","F","SHIP"
   END    trailer
```

Figure A-2. Example of Output Sent to a Client Application

Notice the additional lines before and after the query output. These lines are part of the output protocol, which is fully described in Section 3.8.2. The lines beginning with a period form the *output wrapper*, which contains information pertaining to the addressing of the message and not its contents. The other lines are the output of the *isum-01* query. When we examine this output, we notice that the ship types are missing. We will have to correct the *isum-01* query. To do this, we follow these steps:

Step 1	Choose Windows Queries. This creates a Queries window.
Step 2	Select the *isum-01* query.
Step 3	Choose Commands Edit. This creates a Queries(Edit) window.
Step 4	Insert "PLATID.3" into the SELECT clause in the Query input field.
Step 5	Press VALIDATE to check for syntax errors.
Step 6	Press OK when all errors have been corrected.
Step 7	Press Close to make the Queries window vanish.

To test our correction, we will want to try running the *isum-01* query again. We can make the parser reprocess the sample TACELINT message with the following steps:

Step 1	Choose Windows Message Repository. This creates a Message Repository window displaying all of the incoming messages saved.
Step 2	Select the sample TACELINT message. (If there is more than one message, match the message ID, date/time group, and originator to the output header lines.)
Step 3	Press the Submit button. This causes the processor to reprocess the selected message as if it had just arrived. The processor will overwrite the old *isum.in* file with the new results.

This time the message processor produces the correct output, as shown below:

.MSGID TACELINT
.FROM RHDIAAA
.TO DHDIAZZ/SYJ
.DTG 011200Z APR 93
.ERRORS 0
"21400","010951Z","LS:435244N0751820W","DD","CUSHING","F","SHIP"
"21400","010951Z","LS:435244N0751820W","DD","BURKE","F","SHIP"
"21400","010951Z","LS:435244N0751822W","DD","CUSHING","F","SHIP"
"21400","010951Z","LS:435244N0751822W","DD","BURKE","F","SHIP"
"21401","010951Z","LS:435250N0751850W","DD","LEFTWICH","F","SHIP"
.END

The last step is to correct the routing so that the query output actually goes to ISUM instead of being saved in a file. To do this, we follow these steps:

Step 1 Choose Windows Routing

Step 2 Select TACELINT messages, then press Applications.

Step 3 Select the "cat>isum.in" line, then choose Commands Delete.

Step 4 Press the OK button. This removes the routing entry.

Step 5 Now choose Commands New and make a new routing entry. We follow the same steps as on page, except this time we enter "/usr/isum/isum" into the Application input field.

The registration of ISUM has been tested and completed. As a final test, we can return to the Message Repository window and submit the sample message one more time. This time, the query output is routed to the ISUM program. When ISUM prints its reports, the signals reported in the sample message will be part of its output.

3.4 Registering ISUM with a Client Configuration File.

Client registration with the user interface is good for development and testing, but it is not a practical way to register client systems in the operational environment. We can hardly expect operators in the field to go through the process every time they start the processor.

Client configuration files are the practical way to register clients in the operational world. The client configuration file contains all of the queries and routing table entries necessary to register a

client with the processor. Developers of a client system write the configuration file, which is then distributed with their system software. In the field, operators simply copy this file into the parser configuration directory. When the processor coldstarts, it reads configuration files from all of the systems it serves; afterwards, it has all the information it needs to extract information from incoming messages and route it to the client programs.

A client configuration file is composed of a series of entries. Here is a very brief description of the kinds of entries. A complete description of the syntax of configuration files is presented in Section 3.8.1.

a. A query entry defines and names a single query.
b. A *routing* entry defines a single routing table entry.
c. A *message* entry specifies whether to save messages of a particular type.

In order to register ISUM, we must create a client configurationfile containing the following three entries:

```
query isum-01=
SELECT SOI.1,SOI.2,EMLOC.3,
PLATID.3,PLATID.5,EMLOC.2,PLATID.2
WHERE EMLOC.2="F" AND PLATID.2="SHIP"
FORMAT TABLE
VALIDATE ALL;

route tacelint query isum-01 host localhost
cmd/usr/isum/isum;

set tacelint save=yes;
```

We name this file "isum.ccf" and place it in the configuration directory. During the next coldstart, it will process this file. This will create the *isum-01* query, arrange to run this query on TACELINT messages, route the output to ISUM, and cause the processor to save TACELINT messages. That is everything required to register ISUM.

3.5 Routing Several Message Types to a Single Application.

It is possible that an application might want information from more than one message type. For example, ISUM might want to extract fields from ELINT Requirement Tasking Messages (ERTM) as well as from TACELINT messages. To arrange this, we repeat the registration process: we first write a new query to extract fields from ERTM messages, then route ERTM

messages through this new query to ISUM. We do not have to write a separate client configuration file to do this; instead, we can simply insert the new information into the existing *isum.ccf* file. For example, we might append the following lines to *isum.ccf*:

```
query isum-02=
SELECT AREAREQ.1,AREAREQ.2,TRCPLOT.1,TRCPLOT.2
FORMAT TABLE
VALIDATE ALL;

route ertm query isum-02 host localhost
cmd/usr/isum/isum;
```

Afterwards, ISUM will receive selected fields from both TACELINT and ERTM messages.

3.6 Routing a Single Message Type to Several Applications**.**

Suppose that an application named SIGNALS on our system also needs information from incoming TACELINT messages. We would register it in the same way: first write a query, then route TACELINT messages to it. When a new TACELINT message arrives, the processor will run the *isum-01* query and route the output to ISUM, and then run the query for SIGNALS and route the output to it (or possibly the other way around; the order of evaluation cannot be specified).

It is wise to write a separate client configuration file for each separate application. Some sites might run ISUM but not SIGNALS; others might run SIGNALS but not ISUM. With separate configuration files, each site just puts the configuration files for the applications it runs into the configuration directory. For the same reason, it is also wise to make the individual configuration files completely independent of one another. Although it is possible to define a query in one file and refer to it in another, this should be avoided.

4.0 INSTALLATION, CONFIGURATION, AND OPERATION

This section is a guide for the system administrator at your site. It describes how to install the CMP from the distribution tape, how to customize it for your system, and how to operate the message processor and user interface programs. We assume that the system administrator is familiar with the typical details of UNIX system administration: creating new accounts, operating the tape drive, etc. If not, the *Sun System & Network Manager's Guide* is a good introduction.

4.1 Installation (NOTE: Information located in DII COE “SegDescrip” files)

To facilitate ease of installation, the Army has created “script files” which performs installation and creation of the required paths. Use of the script files reduces the need to define step by step installation procedures.

Once the tar format tape has been extracted onto the system, a number of directories are created (for the binary distribution). These include bin, lib, src, man, and spool. Only bin and lib are important in the actual CMP execution. The directories are described as follows:

NOTE: In this section there are files using the term "jmps" interchangeably with the term CMP. This will be changed in later versions of the CMP software release.

bin: Contains the CMP executable.

lib: Contains the jmps.ini initialization parameter file example, the jmpsrc preference file example (copy to HOME as .jmpsrc), and the selected or available messages along with bitmaps for the program icon display.

src: Contains the jogs compiler for those who need to create binary (.i) message files for CMP.

man: A manual directory currently empty

spool: Output directory currently empty.

4.1.1 CMP Installation (NOTE: Not necessary for DII COE)

Log in as the defined user account and extract the CMP distribution by typing: $ tar xvf /dev/rmt/0m (or your DAT device name.)

4.1.1.1 Journal Installation

1. Log in as any user. cd ~JOURNALING
2. Execute the INSTALLJOURNAL script file.
3. This script will configure the paths to the primary and secondary storage location for messages.
4. cd to the bin directory under JOURNALING and type either
   jnl_server_trarc_fb master (for the file based implementation) or
   jnl_server_trarc_db master (for the database implementation.)
5. You should see the following response:

Initializing msg.db, data.db, memo.db, destInfo.db, hdrInfo.db ..DONE!
---- send log thread created ----------

Listening for Requests ...

4.1.1.2 DISCRIMINATOR Installation

1. Login as any user. cd ~DISC
2. Execute the INSTALLDISC script file.
3. This script will configure the paths in the config.dat file.
4. cd to the bin directory under DISC and type either
   disc_trarc_fb (for the file based implementation) or
   disc_trarc_db (for the database implementation.)

4.1.1.3 JMPS Installation

1. Login as any user. cd ~JMPS
2. Execute the ./lib/JMPinst script file.
3. This script will:
   -Configure the jmps.ini file.
   -Configure all the executables using confadm.
4. cd to the bin directory under JMPS and verify that the path to the jmps.ini file is correct by doing:
   confadm -q jmps
5. If the path is not correct, set the location of the jmps.ini file by doing:
   confadm -s <your path to the file>/JMPS/lib/jmps.ini jmps
6. Set the DISPLAY environment variable to the current display if it is not already set by doing:
   $ export DISPLAY = netmon:0.0 (example for Korn shell)
   $ setenv DISPLAY netmon:0.0 (example for C shell)
7. To run JMPS, cd to the bin directory under JMPS and type:
   $ jmps tmp
   where “tmp” is any file name which will contain the edited message.
8. If “tmp” is a new file, the user will be prompted with a message type to edit based on the name.db file. If the message has already been created

and is to edited, the main jmps input screen will appear.

4.1.1.4 JMAPS Installation

1. Login as root. cd ~JMAPS
2. Execute the ./installation script file. This should configure JMAPS to be run from the jmapsadm account. Answer the prompts with the correct name of the account and where the JMAPS executables have been restored.
3. This script will also:
   -Create the .Mapsconfig file under JMAPS.
   -Configure the binaries using the confdir prog.
   -Install the JMAPS Xdefaults file in the app-defaults directory.
4. To test, coldstart JMAPS by typing:
   $ jmaps coldstart ui

4.1.1.5 CUI Installation

1. Login any user. cd ~CUI
2. Execute the INSTALLCUI script file.
3. This script will ensure that a soft link exists to your current location of jmps (assuming that JMPS has been installed at the same tree level.)
4. cd to the bin directory under CUI and type either
   cui_trarc_fb (for the file based implementation) or
   cui_trarc_db (for the database implementation.)
5. The CUI window will appear.

Note: Requires Journaling Server be installed prior to installation of this module.

4.2 Choose Names and Locations.

You must choose names for the system administrator account and the CMP group. The standard names for these are:

cmpadm	the system administrator user name
cmp	the CMP group name

These accounts are used by the processor for authorization and authentication purposes. The **cmpadm** user is the only user permitted to start and stop the message processor. Other users may run the user interface if they are members of the CMP group.

You must also choose a location for the home directory. This will be the home directory of the system administrator account, and will contain all of the executable and data files. The standard location is **/usr/parser**.

The file system containing the home directory must have at least 20 megabytes of free space for the executable and data files. In addition, more storage will be required in this file system during execution: as messages arrive, they are saved in the message journal; query reports are generated; and data is stored in the system logs. You should treat the above space requirements as an absolute minimum and allocate as much additional space as possible.

It is recommended that you use the standard names and locations, but you can choose others if necessary. If, for example, you decided to use **/usr** as the home directory, then for the remainder of this section you would have to replace every occurrence of **usr/parser** with **/usr**, **/usr parser/bin** with **/usr/new/bin**, etc.

4.2.1 Create User and Group Accounts.

You can create the **CMP** group account by making an entry in the **/etc/group** file. You may choose any group-ID number as long as it is not already used by another group.

You create the **parseradm** user account by making an entry in the **/etc/passwd** file. Make sure that this user is a member of the processor group. Set the user's home directory to **/usr/local/ parser**. (Create this directory if it does not already exist.) We recommend that you use **/bin/csh** as the login shell for this user.

(Complete instructions for adding a new user account are supplied in Chapter 6 of the *System & Network Manager's Guide* volume of the SunOS documentation).

4.3 Configuration

The parser keeps its site configuration information in a plain-text file called **.MAPSconfig** in the administrator's home directory (**/usr/local/CMP**). Almost all of this information is automatically set by the installation process. However, there are eight parser configuration parameters which may need to be adjusted by the parser administrator. These parameters are as follows:

a. **ATT.confdir**: This is the name of the directory containing the c*lient configuration files* (see Section 3.8.1). On startup, the parser processes all of the client configuration files found here. The default value is **/usr/local/parser/config**.

b. **MIO.inputPort**: This is the *tty* port set up for use by the processor. If you are sending messages to the processor over a serial line, then you may need to change this parameter. The default value is **ttya**.

c. **MIO.printEnable**: This tells the processor to print incoming messages as they are received from the serial communications port. A value of 1 (one) enables printing, and 0 (zero) disables printing. The default value is **0**.

d. **MIO.printCmd**: This is the command used to print ordinary text received from the standard input. The print command must be enclosed in double quotes if it includes any spaces. The default value is **lpr**.

e. **CPP.maxSectionWait**: This is the value, in hours, that the processor will wait for missing message sections to arrive. The timer starts running when the first message section is received. When the timer expires, the processor will put the incomplete message in the incomplete MsgDir. There is a separate timer for each message. The default value is 24.

f. **SHMALLOCATOR.size**: This is the number of bytes of shared memory that the processor will use. The default value is **1572864**, which is approximately 1.5 megabytes of shared memory. The remaining shared memory in the system is reserved for other processes. You may increase this value to give the processor more shared memory. If you do this, you may also have to reconfigure your kernel to change the maximum shared memory size. (See Section 3.9.1.1).

g. **IPCS.fieldsOnDisk**: This switch controls how the processor will transfer a message between its internal components. If the value is **0**, it transfers messages using shared memory. This is faster, but may require more shared memory than is available. If the value is **1**, the processor transfers messages as a disk file. This is slower but requires much less shared memory. The default value is **1**.

h. **STANDARD**: This variable controls which message standards a particular process will use. The first character in this variable corresponds to the use of the USMTF standard and the second character corresponds to the use of the OTH Gold standard. The use of a standard is represented by '1' while omission of a standard is represented by '0'.

The configuration parameters in the **.MAPSconfig** file may be changed using any text editor. Changes will take effect the next time the processor is started.

All of the processor programs know the location of the **.MAPSconfig** file because it is build into them as a constant string. This means that if the **.MAPSconfig** file is ever moved, the executable files must be updated with the new location. This could happen if, for example, the entire processor tree must be moved from one file system to another.

Suppose, for example, that we have just moved the parser directory tree to **/export/parser**. Then, to update the executables, we must execute the following commands:

```
cd/export/parser
bin/confdir-s/export/parser bin/jmaps
```

This will update all of the system and demonstration programs. There will be several "*progname* is not a configurable executable file" warning messages, which should be ignored.

4.4 Operation

In general, only the administrator can directly control the message processor. Other users can only run the user interface program. The commands associated with controlling the parser message processor are as follows:

coldstart: run the message processor for the first time
snapshot: save the internal state of the message processor
shutdown: stop the message protocol
warmstart: restart the message processor from a previously- saved state
killprocessor: terminate the message processor without saving the current state.

4.4.1 Initializing the Processor.

The **coldstart** command is used when the CMP has never been run on your computer before (i.e., the first time it is started). It initializes the system and creates internal data structures necessary for proper operation. Afterwards, the processor will read incoming messages, but it will not perform any message processing until it is told about its client applications.

By default, the processor will not attempt to read messages from a serial communication line. If you want it to listen to the serial communications line, use the command **coldstartcomm** instead.

When you run **coldstart** you create an "empty" processor, with no messages received and no queries or routing information entered. The results of previous operations will be lost. If you wish to resume a previous execution instead of starting a new processor "from scratch," you must use the **warmstart** command instead.

4.4.2 Saving the System State.

The **snapshot** command records the internal state of the message processor. This state information is complete in that it includes the message journal, query reports, the error log, a description of the processor client applications, and the message information these clients require.

The state information is kept in the** storage **directory in a file that is always named** data**. This file is overwritten by each snapshot or shutdown operation. A single backup copy of the data file is always saved before creating a new version. This file is named** dataBack**.

4.4.3 Shutting Down the Processor.

The **shutdown** command is performed whenever it is necessary to turn off the processes. A shutdown performs a snapshot before terminating the processes. This allows the administrator to warmstart at a later date, and the system will pick up where it left off.

When it is not running, new messages that arrive at the communication port will not be read. They will be lost.

4.4.4 Restarting the Processor.

The **warmstart** command starts the processor, recovering the system state from the **data** file created by a previous **snapshot** or **shutdown**.

By default, it will not attempt to read messages from a serial communication line. If you want it to listen to the serial communications line, use the command w**armstartcomm** instead.

4.4.5 Killing Processes.

The **kill** command terminates the message processor and all user interface processes running on the local machine and removes all shared memory segments and semaphores that belong to the user. This command does not save the current state.

5.0 Normalization Installation and Initialization.

The first step in using the normalization software is to customize the datafiles for use with the host application software. Please refer to the paragraph titled "Customizing the Normalization Datafiles" for detailed information on this process.

Next, the user must modify the "INSTALL_DIR" variable in the tem_env file located in the installation directory "environment" subdirectory. It should be changed to reflect the directory into which the Normalization software was installed. After modifying the tem_env file from within the C-Shell, the user should source the tem_env file as follows: "source tem_env". There should be no error messages associated with executing the source command.

To use the normalization function, the calling application must first invoke the initialize() function. Its calling sequence is "int initialize();". It returns SUCCESS on successful initialization and returns FAILURE on unsuccessful initialization. SUCCESS and FAILURE are defined in the

"normaliz.h" header file found in the install directory "include" subdirectory. The initialize function reads in the lookup tables found in the install directory "datafiles" subdirectory. It should be called only once at application start-up.

5.1 NORMALIZATION Installation

1. Login as any user. cd ~NORMALIZATION
2. Execute the INSTALLNORM script file.
3. This script file will configure all necessary normalization environment variables.
4. cd to the bin directory under NORMALIZATION and type $set_platform.

5.2 Termination.

The Normalization software is a callable library function. It is not an independent process. Use of the Normalization software places no special termination constraints on the calling application.

5.3 Restart.

Upon restarting of the calling application, the initialize() function needs to be reinvoked.

5.4 Outputs.

The outputs of the Normalize function are the returned error code (SUCCESS or FAILURE), indicating successful normalization or unsuccessful normalization) and the output variable containing the converted data. The converted data will always be in character string format. The character string format is appropriate since most calling applications will be databasing the information (in which case a character string is appropriate for the insert SQL statement) or putting the data into an outbound message. If however, the calling application requires integer and float type values, it is possible to convert the character string output to the desired format via "sscanf()" or "atoi()", which are part of the standard "C" library.

5.5 Error Messages

The following is a listing of the error messages output by the Normalization software, the associated meaning of the message, and the action to be taken when each message appears;

Error Message: "initialization() must be called prior to invoking normalization function"

Meaning: The initialization function has not been invoked yet.

Action: Add a call to the initialization function as described in paragraph 5.2.3.1 above.

Error Message: "conversion failed"

Meaning: The Normalization software was unable to convert the specified input.

Action: Examine input parameter and all other parameters to verify that a corresponding match exists in the appropriate datafile. Refer to the Appendix titled "Customization of the Datafiles" for information on the datafiles and their usage.

Error Message: "Please set environment variable WORKING_DIR =location of datafiles"

Meaning: The Normalization software is looking for the location of the datafiles during initialization.

Action: Verify that the tem_env file has been modified correctly and that the invoking shell has sourced the tem_env files.

Error Message: "counting data files failed"

Meaning: The Normalization software attempts to count the number of datafiles in order to allocate memory for the data tables. This message indicates that the software is unable to read the datafiles.

Action: There could be two possible causes for this error message. One, check the directory and file permissions of the location of the datafiles. The user id assumed by the calling application must have read permission on the datafiles and execute permission on the directory containing the datafiles. Second, the environment variable WORKING_DIR should be set to the location of the datafiles directory. Check the tem_env file to verify that it is set correctly.

Error Message: "opening data file failed"

Meaning: The Normalization software is unable to open one or more of the datafiles.

Action: Check the directory and file permissions of the location of the datafiles. The user id assumed by the calling application must have read permission on the datafiles and execute permission on the directory containing the datafiles.

Error Message: "load row failed"

Meaning: The Normalization software was unable to load the data properly from the datafiles.

Action: Verify that the datafiles are in the format specified in the Appendix titled "Customization of the Datafiles".

Error Message: "error in cur_bfa entry: datafile file_number, row # entry_number"

Meaning: The Normalization software was unable to load the data properly from the datafiles.

Action: Verify that the datafiles are in the format specified in the paragraph titled "Customization of the Datafiles." In particular, check the cur_bfa column (column 1) and the row number specified in the error message. The row numbers start with valid data rows and do not include comment lines.

Error Message: "error in other bfa entry: datafile file_number, row # entry_number"

Meaning: The Normalization software was unable to properly load the data from the datafiles.

Action: Verify that the datafiles are in the format specified in the paragraph titled "Customization of the Datafiles." In particular, check the other_bfa column (column 2) and the row number specified in the error message. The row numbers start with valid data rows and do not include comment lines.

Error Message: "error in in_or_out entry: datafile file_number, row # entry_number"

Meaning: The Normalization software was unable to load the data properly from the datafiles.

Action: Verify that the datafiles are in the format specified in the paragraph titled "Customization of the Datafiles." In particular, check the in_or_out column (column 3) and the row number specified in the error message. The row numbers start with valid data rows and do not include comment lines.

Error Message: "error in rpc_or_msg entry: datafile file_number, row # entry_number"

Meaning: The Normalization software was unable to load the data properly from the datafiles

Action: Verify that the datafiles are in the format specified in the paragraph titled "Customization of the Datafiles." In particular, check the rpc_or_msg column (column 4) and the row number specified in the error message. The row numbers start with valid data rows and do not include comment lines.

Error Message: "error in msg_kind entry: datafile file_number, row # entry_number"

Meaning: The Normalization software was unable to load the data properly from the datafiles.

Action: Verify that the datafiles are in the format specified in the paragraph titled "Customization of the Datafiles." In particular, check the msg_kind column (column 5) and the row number specified in the error message. The row numbers start with valid data rows and do not include comment lines.

Error Message: "couldn't find match"

Meaning: The Normalization server was unable to find a matching output string for the given input string.

Action: Examine the input parameter and all other parameters to verify that a corresponding match exists in the appropriate datafile. Refer to the paragraph titled "Customization of the Datafiles" for information on the datafiles and their usage.

Error Message: "geometry lookup failed"

Meaning: The Normalization server was unable to find a matching output geometry type for the given parameters. Note that the only valid geometry types are MGRS, LAT_LONG, and EAST_NORTH.

Action: Examine all parameters except the input and output parameters, and verify that a corresponding match exists in the appropriate datafile. Refer to the paragraph titled "Customization of the Datafiles" for information on the datafiles and their usage.

Error Message: "non supported geometry type"

Meaning: The geometry types contained in the datafile are not supported types. Note that the only valid geometry types are MGRS, LAT_LONG, and EAST_NORTH.

Action: Refer to the paragraph titled "Customization of the Datafiles" for information on the datafiles and their usage.

Error Message: "time lookup failed"

Meaning: The Normalization server was unable to find a matching output time type for the given parameters. Note that the only valid time types are DTG and SECS.

Action: Examine all parameters except the input and output parameters, and verify that a corresponding match exists in the appropriate datafile. Refer to the paragraph titled "Customization of the Datafiles" for information on the datafiles and their usage.

Error Message: "non supported time type"

Meaning: The time types contained in the datafile are not supported types. Note that the only valid time types are DTG and SECS.

Action: Refer to the paragraph titled "Customization of the Datafiles" for information on the datafiles and their usage.

Error Message: "conversion of quantity for <%s> failed"

Meaning: The Normalization software was unable to convert the input quantity.

Action: Examine input string to verify that its data is valid.

Error Message: "no data files found in WORKING_DIR"

Meaning: The Normalization software was unable to locate the datafiles.

Action: Verify that the tem_env file has been modified correctly and that the invoking shell has sourced the tem_env files.

5.6 Customization of the Datafiles

The Normalization software reads input from datafiles in the "WORKING_DIR" directory. The datafiles are located in the install directory subdirectory "datafiles." The following data files are customarily used by the Normalization software:

datafile0 : datafile for normalization of coordinates
datafile1 : datafile for normalization of time conversions
datafile2 : datafile for normalization of primary option
datafile3 : datafile for normalization of resource class
datafile4 : datafile for normalization of units
datafile5 : datafile for normalization of geometries
datafile6 : datafile for normalization of special skills
datafile7 : datafile for normalization of quantities

All conversions are done through table lookup except the coordinate conversions, time conversions, and quantity conversions.

These data files need to be modified to suit the specific requirements of the calling application. They may be modified using any ASCII text editor. It is not necessary to recompile the code after modification of the data files; however, the calling application must call the initialize function prior to using the normalization function. The initialize function reads in the values from the data files into internal lookup tables.

Additional data files may be added for conversions not covered by these data files. Simply create another data file in the same directory as the previously listed data files. The new data files should be numbered starting at "datafile8" and the existing data files should be used as templates. It is not necessary to recompile the normalization code after addition of datafiles. Simply reinvoke the initialization function from the calling application.

5.7 Datafile Format.

The entries in datafile0 and datafile1 are listed in the following format:

CUR BFA	OTHER BFA	IN/OUT	RPC/MSG	MESSAGE NUMBER	INPUT TYPE	OUTPUT TYPE

Example:

MCS CSSCS IN RPC S501 MGRS LAT_LONG

The CURRENT BFA is the one running the normalization server.

The OTHER BFA is the one that will be a recipient of an outgoing message or the sender of an incoming message.

The IN/OUT column is used to indicate whether the conversion applies to inbound data (to be databased) or outbound data (for autofill).

The RPC/MSG column indicates whether the medium for transferring the data is Remote Procedure Call (RPC) or via USMTF/ATCCS messages.

The MESSAGE NUMBER is used to further refine the type of conversion to be applied to the data.

The INPUT type column indicates what the input geometric type to the normalization routine will be.

The OUTPUT type is the desired returned type.

NOTE: The only valid input and output types for geometric conversions are: MGRS, LAT_LONG, and EAST_NORTH. MGRS is in format CCRSsEEEENNNN where CC is grid zone column, R is grid zone row, S is UTM 100,000 meter sq. column, s is UTM 100,000 meter sq. row, EEEE is UTM 10 meter easting, NNNN is UTM 10 meter northing.

EAST_NORTH is one meter Easting and Northing, and LAT_LONG is latitude and longitude.

The entry NA may be used in any column (except INPUT and OUTPUT) to indicate that the input and output pair applies to any entry in the corresponding column.

The CURRENT BFA entry must start in the first column, i.e., no additional spaces before it. To insert a comment, begin the line with '##'.

For time conversions, the only valid entries for input and output type are DTG and SECS

The other data files, datafile2 through datafile7, contain entire lookup tables. All of the columns are the same as the time and geometric conversion data files except the INPUT and OUTPUT columns. The INPUTS column indicates the input string to the normalization function. The OUTPUT column indicates the corresponding output string for the given input string.

APPENDIX B
MESSAGE DATA TABLES

1. SCOPE

This appendix defines the various message standards supported by CMP and provides a listing of currently available tables.

1.1 MESSAGE GENERATION DATA TABLES :

Directory of data tables : /h/DTJMPS/data

FILE NAME	NUMBER OF MESSAGES	DESCRIPTION OF MESSAGES
gold/	18 Msgs :	OTH GOLD
mts/	140 Msgs :	MTS VMF
tf21/	44 Msgs :	9 ACCS, 11 JOINT USMTF.93, 1 NATO 5 CSSCS, 5 VMF, 8 NBC, 5 MISC
tf21-vmf/	113 Msgs	39 TF21 & 74 USA VMF
usmtf93/	248 Msgs :	JOINT USMTF.93
usmtf95/	284 Msgs :	JOINT USMTF.95
usmtf95-vmf/	358 Msgs :	284 JOINT USMTF.95, 74 USA VMF
usmtf97/	297 Msgs :	JOINT USMTF 97
vmf/	74 Msgs :	74 USA VMF

Each of the above directories will contain the following binary files :

cod.i, frn.i, frndx.i, des.i, exp.i, hlp.i, msg.i msgndx.i, set.i, setndx.i, std.s, usmtf.db

1.2 MESSAGE PARSER DATA TABLES :

Directory of data tables : /h/DTJMAPS/data

FILE NAME	NUMBER OF MESSAGES	DESCRIPTION OF MESSAGES

jgold/	8 Msgs	OTH GOLD
jmts/	140 Msgs :	MTS VMF9 ACCS,
jtf21/	44 Msgs	11 JOINT USMTF.93, 1 NATO 5 CSSCS, 5 VMF, 8 NBC, 5 MISC
jtf21-vmf/	113 Msgs	39 TF21 & 74 USA VMF
jusmtf93/	248 Msgs :	JOINT USMTF.93
jusmtf95	284 Msgs	JOINT USMTF.95
jusmtf95-vmf/	358 Msgs	284 JOINT USMTF.95, 74 USA VMF
jusmtf97/	297 Msgs	JOINT USMTF 97
jvmf/	74 Msgs	74 USA VMF

Each of the above directories will contain the following binary files :

mtfcod.i, mtffrn.i, mtffrndx.i, mtfdes.i, mtfexp.i, mtfhlp.i, mtfmsg.i mtfmsgndx.i, mtfset.i, mtfsetndx.i, std.s, usmtf.db

The 16 object validation tables are grouped into two directories, 8 in the /h/DTJMPS/data directory for the message generation modules and 8 in the /h/DTJMAPS/data directory for message parsing. To select the appropriate message table of interest the system operator must edit the CMP configuration files.

The procedure for JMAPS is :

a. cd /h/JMAPS/Scripts

b. Edit configuration file vi .MAPSconfig

c. change DTAB.dir entry path to /h/DTJMAPS/data created_jmaps_object_tables

d. Save .MAPSconfig

The procedure for JMPS is :

a. cd /h/JMPS/lib

b. Edit configuration file jmps.ini

c. Change DTABS.dir entry path to /h/DTJMPS/data created_jmps_object_tables

d. Save jmps.ini